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 № 2

по дисциплине «Криптографические методы защиты информации»

Современные симметричные шифры

Студент гр. БИБ224
Г. А. Симаков
«20» марта 2024 г.

Руководитель
Заведующий кафедрой информационной безопасности киберфизических систем
канд. техн. наук, доцент
_____О.О. Евсютин
«___» _____ 2024 г.

Москва 2024

СОДЕРЖАНИЕ

1 Задание на практическую работу

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

В рамках практической работы необходимо выполнить следующее:

1) написать программную реализацию одного из следующих симметричных шифров (по выбору студента):
   - Магма;
   - Кузнечик;
   - AES;
2) подготовить отчет о выполнении работы.

Программа должна обладать следующей функциональностью:

1) принимать на вход файл, содержащий открытый текст, подлежащий зашифрованию, или шифртекст, подлежащий расшифрованию;
2) принимать на вход секретный ключ;
3) [***дополнительная опция, не являющаяся обязательной***] давать пользователю возможность выбирать режим работы блочного шифра;
4) осуществлять зашифрование или расшифрование выбранного файла по выбору пользователя и сохранять результат в новом файле.

Отчет должен содержать следующие составные части:

1) раздел с заданием;
2) раздел с краткой теоретической частью;
3) раздел с описанием программной реализации с учетом особенностей выбранной среды разработки и языка программирования;
4) раздел с результатами работы программы;
5) раздел с выводами о проделанной работе.

1)

2 Теория

Основное отличие между современными и историческими (классическими) шифрами, к последним из которых относятся все алгоритмы шифрования докомпьютерной эпохи, заключается в способе представления данных. Современные шифры обрабатывают данные в цифровом виде как битовые последовательности, природа которых не имеет значения, в то время как исторические шифры могли работать лишь с текстами, что накладывало определенные ограничения на использовавшийся математический аппарат.

При этом основные криптографические преобразования остались прежними, лишь усложнился способ их реализации (здесь следует отметить, что речь идет исключительно о симметричных шифрах).

Основным методом криптоанализа исторических шифров является статистический криптоанализ, когда криптоаналитик изучает статистику встречаемости символов в шифртексте и, используя полученную информацию, может осуществить дешифрование зашифрованных данных.

До появления компьютерной криптографии данные, которые необходимо было защищать с помощью шифрования, практически всегда представляли собой текст на естественном языке, обладающем высокой избыточностью. Именно высокая избыточность открытых текстов является причиной успешного применения различных статистических атак для взлома классических шифров. В связи с этим К. Шенноном были предложены два основных криптографических метода сокрытия избыточности открытого текста: перемешивание и рассеивание.

Перемешивание устраняет зависимость между открытым текстом и шифртекстом, затрудняет попытки найти в шифртексте избыточность и статистические закономерности. Перемешивание осуществляется с помощью подстановок (замен).

Рассеивание перераспределяет избыточность открытого текста, распространяя ее на весь шифртекст. Каждый бит открытого текста при шифровании должен повлиять на максимальное число других бит открытого текста. Простейшим способом создания рассеивания является перестановка.

По отдельности ни перемешивание, ни рассеивание не могут обеспечить надежного шифрования, поэтому классические подстановочные и перестановочные шифры неприменимы в современном мире, однако криптосистемы, в которых реализованы оба этих метода, обладают высокой криптостойкостью.

Можно выделить следующие основные элементарные операции над данными, объединение и многократное повторение которых положено в основу всех современных симметричных алгоритмов шифрования:

- замена элементов данных (короткие битовые последовательности) с помощью специальных таблиц;
- перестановка элементов данных;
- битовый сдвиг;
- операции сложения/вычитания по модулю;
- поразрядное сложение по модулю 2.

Ниже перечислены основные критерии оценки симметричных алгоритмов шифрования (в частности, данные критерии использовались при выборе стандарта шифрования AES).

1. Реальная защищенность от криптоаналитических атак. При этом основными методами криптоанализа являются:
- дифференциальный криптоанализ;
- расширения для дифференциального криптоанализа;
- поиск наилучшей дифференциальной характеристики;
- линейный криптоанализ;
- интерполяционное вторжение;
- вторжение с частичным угадыванием ключа;
- вторжение с использованием связанного ключа;
- вторжение на основе обработки сбоев;
- поиск лазеек.

2. Статистическая безопасность криптографического алгоритма.

3. Надежность математической базы криптографического алгоритма.

4. Расчетная сложность криптографического алгоритма для программной и аппаратной реализаций (может оцениваться скоростью преобразований).

5. Требования к памяти при программной и аппаратной реализациях. При аппаратной реализации оценивается числом логических элементов, при программной – количеством необходимой оперативной и постоянной памяти, в том числе для различных платформ и сред.

6. Гибкость алгоритма, то есть:
- возможность работы с иными длинами начальных ключей и информационных блоков;
- безопасность реализации в широком диапазоне различных платформ и приложений, включая 8-битовые процессоры;

- возможность использования криптографического алгоритма в качества поточного шифра или генератора псевдослучайных чисел, алгоритма хеширования, для обеспечения подлинности сообщений (выработка кодов аутентичности сообщений) и т.п.;
- одинаковая сложность как аппаратной, так и программной реализации, а также программно-аппаратной реализации.

Современная криптография выделяет два типа симметричных криптосистем: блочные и поточные.

Поточные шифры преобразуют открытый текст в шифртекст последовательно по одному биту путем сложения битов открытого текста с битами гаммы по модулю 2. Гамма необходимой длины формируется из секретного ключа шифрования фиксированной длины. При этом поточный шифр может быть построен на основе блочного шифра при использовании того в специальном режиме.

Блочные шифры обрабатывают открытый текст, разбивая его на блоки равного размера, причем все современные блочные шифры являются раундовыми. Это означает, что зашифрование одного блока открытого текста (расшифрование одного блока шифртекста) заключается в многократном применении к нему некоторой последовательности элементарных криптографических операций. Соответственно, однократное применение такой последовательности к блоку данных называется раундом.

За годы, прошедшие с момента создания DES, он подвергался многочисленным исследованиям и попыткам криптоанализа. Были разработаны такие методы криптоанализа как дифференциальный и линейный, проведены успешные атаки на некоторые варианты DES с меньшим числом циклов. Но в целом все проведенные исследования подтвердили надежность DES. Поэтому главным его недостатком стала длина ключа. Быстрое развитие средств вычислительной техники уже в первой половине 90-х годов XX века сделало возможным полный перебор всех ключей DES за реальное время.

В связи с этим в 1997 году НИСТ объявил конкурс на создание нового общенационального стандарта шифрования данных, который должен был прийти на замену DES. На конкурс были представлены 15 алгоритмов симметричного шифрования из разных стран, из которых затем отобрали 5 финалистов. В итоге лучшим был признан шифр Rijndael, разработанный двумя бельгийскими криптографами Винсентом Рэйменом и Йоаном Дайменом. В 2001 году опубликован стандарт FIPS PUB 197, содержащий улучшенный стандарт шифрования данных AES (Advanced Encryption Standard), основанный на Rijndael.

Одним из требований к новому стандарту была возможность работы с информационными блоками и ключами шифрования различной длины. Поэтому Rijndael

был сконструирован как шифр с переменной длиной блока и переменной длиной ключа, которые независимо могут приниматься равными 128, 192 и 256 бит. AES представляет собой усеченную версию шифра Rijndael с варьируемой длиной ключа 128, 192 или 256 бит, но с постоянной длиной блока 128 бит.

AES является итерационным шифром, что означает, как уже было сказано, многократное повторение основного криптографического преобразования. При этом число раундов не фиксировано, а зависит от длины ключа: 10 раундов для ключа длиной 128 бит, 12 раундов для ключа длиной 192 бита и 14 раундов для ключа длиной 256 бит.

3 Программная реализация

Программная реализация поделена на несколько частей:

1. Импортирование встроенной библиотеки binascii, инициализация констант
2. Матричные операции, преобразования строк в матрицы и наоборот и операцию XOR.
3. Программная реализация функций subBytes(), invSubBytes(), shiftRows(), invShiftRows(), mixColumns(), invMixColumns(), keyExpansion()
4. Функционал шифрования
5. Функционал дешифрования
6. Запуск и выбор режима работы

4 Примеры работы программы

Рисунок 1 Пример 1: английский язык

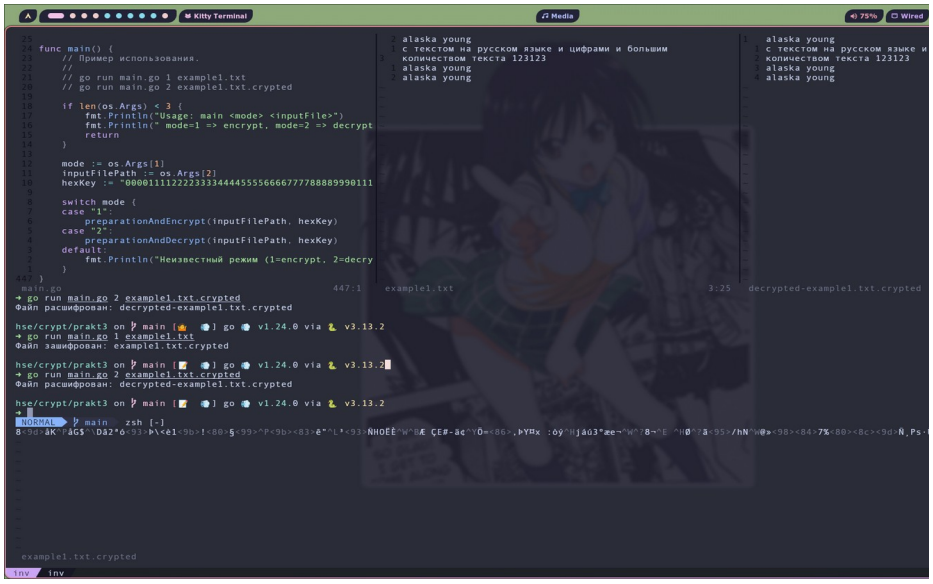

Рисунок 2 Пример 2: удлинённый текст, русский язык и цифры

Рисунок 3 Пример 3: длинный текст на разных языках и со спецсимволами

5 Выводы о проделанной работе

В ходе лабораторной работы мы реализовали процесс шифрования и дешифрования файла с использованием алгоритма AES. Файл был успешно зашифрован, а затем дешифрован обратно с применением ключа, демонстрируя практическое применение алгоритма для защиты данных.

7 Список использованных источников

1. Симметричный алгоритм блочного шифрования Advanced Encryption Standart – 2024. URL: https://habr.com/ru/articles/534620/ (дата обращения 20.03.2024)

2. AES (стандарт шифрования) – 2024. URL: https://ru.wikipedia.org/wiki/AES_(%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) 7114 (дата обращения 20.03.2024)

ПРИЛОЖЕНИЕ А.

Программная реализация

```go
package main

import (
    "encoding/hex"
    "fmt"
    "io"
    "log"
    "os"
)


var sBox = [256]byte{
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16,
}

var invSBox = [256]byte{
    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,
    0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,
    0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,
    0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
    0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
    0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
    0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
    0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
    0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
    0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
    0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
    0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D,
}

var rCon = [18]byte{
    0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
    0x80, 0x1B, 0x36, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00,
}

var keySize = map[int]int{
    16: 10, // AES-128
    24: 12, // AES-192
    32: 14, // AES-256
}
```

```go
type matrix [4][4]byte

func mulBy02(b byte) byte {
    var shifted = b << 1
    if (b & 0x80) != 0 {
        shifted ^= 0x1B
    }
    return shifted
}

func mulBy03(b byte) byte {
    return mulBy02(b) ^ b
}

func mulBy09(b byte) byte {
    return mulBy02(mulBy02(mulBy02(b))) ^ b
}

func mulBy0b(b byte) byte {
    return mulBy02(mulBy02(mulBy02(b))) ^ mulBy02(b) ^ b
}

func mulBy0d(b byte) byte {
    return mulBy02(mulBy02(mulBy02(b))) ^ mulBy02(mulBy02(b)) ^ b
}

func mulBy0e(b byte) byte {
    return mulBy02(mulBy02(mulBy02(b))) ^ mulBy02(mulBy02(b)) ^ mulBy02(b)
}

func bytesToMatrix(src []byte) matrix {
    var m matrix
    for i := 0; i < 16; i++ {
        m[i/4][i%4] = src[i]
    }
    return m
}

func matrixToBytes(m matrix) []byte {
    var out [16]byte
    idx := 0
    for i := 0; i < 4; i++ {
        for j := 0; j < 4; j++ {
            out[idx] = m[i][j]
            idx++
        }
    }
    return out[:]
}

func xorBytes(a, b []byte) []byte {
    n := len(a)
    if len(b) < n {
        n = len(b)
    }
    out := make([]byte, n)
    for i := 0; i < n; i++ {
        out[i] = a[i] ^ b[i]
    }
    return out
}

func switchSBox(arr []byte) []byte {
    for i := 0; i < len(arr); i++ {
        arr[i] = sBox[arr[i]]
    }
    return arr
}
```

```go
func subBytes(state matrix) matrix {
    for i := 0; i < 4; i++ {
        for j := 0; j < 4; j++ {
            state[i][j] = sBox[state[i][j]]
        }
    }
    return state
}

func invSubBytes(state matrix) matrix {
    for i := 0; i < 4; i++ {
        for j := 0; j < 4; j++ {
            state[i][j] = invSBox[state[i][j]]
        }
    }
    return state
}

func shiftRows(state matrix) matrix {
    state[0][1], state[1][1], state[2][1], state[3][1] =
        state[1][1], state[2][1], state[3][1], state[0][1]

    state[0][2], state[1][2], state[2][2], state[3][2] =
        state[2][2], state[3][2], state[0][2], state[1][2]

    state[0][3], state[1][3], state[2][3], state[3][3] =
        state[3][3], state[0][3], state[1][3], state[2][3]

    return state
}

func invShiftRows(state matrix) matrix {
    state[0][1], state[1][1], state[2][1], state[3][1] =
        state[3][1], state[0][1], state[1][1], state[2][1]

    state[0][2], state[1][2], state[2][2], state[3][2] =
        state[2][2], state[3][2], state[0][2], state[1][2]

    state[0][3], state[1][3], state[2][3], state[3][3] =
        state[1][3], state[2][3], state[3][3], state[0][3]

    return state
}

func mixColumns(s matrix) matrix {
    var temp matrix
    for i := 0; i < 4; i++ {
        a0 := s[i][0]
        a1 := s[i][1]
        a2 := s[i][2]
        a3 := s[i][3]

        temp[i][0] = mulBy02(a0) ^ mulBy03(a1) ^ a2 ^ a3
        temp[i][1] = a0 ^ mulBy02(a1) ^ mulBy03(a2) ^ a3
        temp[i][2] = a0 ^ a1 ^ mulBy02(a2) ^ mulBy03(a3)
        temp[i][3] = mulBy03(a0) ^ a1 ^ a2 ^ mulBy02(a3)
    }
    return temp
}
```

```go
func invMixColumns(s matrix) matrix {
    var temp matrix
    for i := 0; i < 4; i++ {
        a0 := s[i][0]
        a1 := s[i][1]
        a2 := s[i][2]
        a3 := s[i][3]

        temp[i][0] = mulBy0e(a0) ^ mulBy0b(a1) ^ mulBy0d(a2) ^ mulBy09(a3)
        temp[i][1] = mulBy09(a0) ^ mulBy0e(a1) ^ mulBy0b(a2) ^ mulBy0d(a3)
        temp[i][2] = mulBy0d(a0) ^ mulBy09(a1) ^ mulBy0e(a2) ^ mulBy0b(a3)
        temp[i][3] = mulBy0b(a0) ^ mulBy0d(a1) ^ mulBy09(a2) ^ mulBy0e(a3)
    }
    return temp
}

func addRoundKey(state, roundKey matrix) matrix {
    for i := 0; i < 4; i++ {
        for j := 0; j < 4; j++ {
            state[i][j] ^= roundKey[i][j]
        }
    }
    return state
}

func keyExpansion(key []byte) []matrix {
    length := len(key)
    if _, ok := keySize[length]; !ok {
        log.Fatalln("Ошибка длины ключа в его расширении:", length)
    }
    Nk := length / 4
    Nr := keySize[length]
    totalWords := 4 * (Nr + 1)

    words := make([][]byte, Nk)
    for i := 0; i < Nk; i++ {
        words[i] = key[4*i : 4*(i+1)]
    }

    i := Nk
    for i < totalWords {
        temp := make([]byte, 4)
        copy(temp, words[i-1])

        if i%Nk == 0 {
            t0 := temp[0]
            temp = temp[1:]
            temp = append(temp, t0)
            temp = switchSBox(temp)
            temp[0] ^= rCon[i/Nk]
        } else if Nk > 6 && (i%Nk == 4) {
            temp = switchSBox(temp)
        }
        out := xorBytes(temp, words[i-Nk])
        words = append(words, out)
        i++
    }

    var roundKeys []matrix
    idx := 0
    for j := 0; j < len(words)/4; j++ {
        var m matrix
        for row := 0; row < 4; row++ {
            copy(m[row][:], words[idx+row])
        }
        roundKeys = append(roundKeys, m)
        idx += 4
    }
    return roundKeys
}
```

```go
func encryptBlock(openText, key []byte) []byte {
    if len(openText) != 16 {
        log.Fatalln("Ошибка на длине блока текста")
    }
    if _, ok := keySize[len(key)]; !ok {
        log.Fatalln("Ошибка на длине ключа при шифровании")
    }

    state := bytesToMatrix(openText)
    roundKeys := keyExpansion(key)
    Nr := keySize[len(key)]

    state = addRoundKey(state, roundKeys[0])

    for r := 1; r < Nr; r++ {
        state = subBytes(state)
        state = shiftRows(state)
        state = mixColumns(state)
        state = addRoundKey(state, roundKeys[r])
    }

    state = subBytes(state)
    state = shiftRows(state)
    state = addRoundKey(state, roundKeys[Nr])

    return matrixToBytes(state)
}

func decryptBlock(cipherText, key []byte) []byte {
    if len(cipherText) != 16 {
        log.Fatalln("Ошибка на длине блока зашифрованного текста")
    }
    if _, ok := keySize[len(key)]; !ok {
        log.Fatalln("Ошибка на длине ключа при расшифровании")
    }

    state := bytesToMatrix(cipherText)
    roundKeys := keyExpansion(key)
    Nr := keySize[len(key)]

    state = addRoundKey(state, roundKeys[Nr])
    state = invShiftRows(state)
    state = invSubBytes(state)

    for r := Nr - 1; r > 0; r-- {
        state = addRoundKey(state, roundKeys[r])
        state = invMixColumns(state)
        state = invShiftRows(state)
        state = invSubBytes(state)
    }

    state = addRoundKey(state, roundKeys[0])

    return matrixToBytes(state)
}
```

```go
func preparationAndEncrypt(inputFilePath, keyHex string) {
    file, err := os.Open(inputFilePath)
    if err != nil {
        log.Fatalln("Не удалось открыть файл:", err)
    }
    defer file.Close()

    data, err := io.ReadAll(file)
    if err != nil {
        log.Fatalln("Ошибка чтения файла:", err)
    }

    outputFileName := inputFilePath + ".crypted"

    blocks := splitIntoBlocks(data, 16)

    last := blocks[len(blocks)-1]
    var counter byte
    for len(last) < 15 {
        last = append(last, 0x00)
        counter++
    }
    if len(last) == 15 {
        last = append(last, counter)
    }
    blocks[len(blocks)-1] = last

    k, err := hex.DecodeString(keyHex)
    if err != nil {
        log.Fatalln("Неверный ключ (hex):", err)
    }

    var result []byte
    for _, blk := range blocks {
        enc := encryptBlock(blk, k)
        result = append(result, enc...)
    }

    err = os.WriteFile(outputFileName, result, 0644)
    if err != nil {
        log.Fatalln("Ошибка записи:", err)
    }
    fmt.Println("Файл зашифрован:", outputFileName)
}
```

```go
func preparationAndDecrypt(inputFilePath, keyHex string) {
    file, err := os.Open(inputFilePath)
    if err != nil {
        log.Fatalln("Не удалось открыть файл:", err)
    }
    defer file.Close()

    data, err := io.ReadAll(file)
    if err != nil {
        log.Fatalln("Ошибка чтения файла:", err)
    }

    outputFileName := "decrypted-" + inputFilePath

    blocks := splitIntoBlocks(data, 16)

    k, err := hex.DecodeString(keyHex)
    if err != nil {
        log.Fatalln("Неверный ключ (hex):", err)
    }

    var decrypted []byte
    for _, blk := range blocks {
        dec := decryptBlock(blk, k)
        decrypted = append(decrypted, dec...)
    }

    if len(decrypted) >= 16 {
        lastByte := decrypted[len(decrypted)-1]
        if lastByte > 0 && lastByte < 15 {
            decrypted = decrypted[:len(decrypted)-1-int(lastByte)]
        }
    }

    err = os.WriteFile(outputFileName, decrypted, 0644)
    if err != nil {
        log.Fatalln("Ошибка записи файла:", err)
    }
    fmt.Println("Файл расшифрован:", outputFileName)
}

func splitIntoBlocks(data []byte, blockSize int) [][]byte {
    var result [][]byte
    for i := 0; i < len(data); i += blockSize {
        end := i + blockSize
        if end > len(data) {
            end = len(data)
        }
        chunk := make([]byte, end-i)
        copy(chunk, data[i:end])
        result = append(result, chunk)
    }
    if len(result) == 0 {
        result = append(result, []byte{})
    }
    return result
}
```

```go
func main() {
    // Пример использования.
    //
    // go run main.go 1 example1.txt
    // go run main.go 2 example1.txt.crypted

    if len(os.Args) < 3 {
        fmt.Println("Usage: main <mode> <inputFile>")
        fmt.Println(" mode=1 => encrypt, mode=2 => decrypt")
        return
    }

    mode := os.Args[1]
    inputFilePath := os.Args[2]
    hexKey := "000011112222333344445555666677778888999011101010"

    switch mode {
    case "1":
        preparationAndEncrypt(inputFilePath, hexKey)
    case "2":
        preparationAndDecrypt(inputFilePath, hexKey)
    default:
        fmt.Println("Неизвестный режим (1=encrypt, 2=decrypt).")
    }
}
```