Министерство науки и высшего образования
Липецкий государственный технический университет

Факультет автоматизации и информатики
Кафедра прикладной математики

Отчет по лабораторной работе № 2
по дисциплине «Безопасности компьютерных систем»
на тему «Симметричные криптосистемы: блочные шифры»

Студент _____ Пестова А.Ю.
подпись, дата фамилия, инициалы

Группа ПМ-19-1

Руководитель
к.т.н., доцент _____ Сысоев А.С.
учёная степень, учёное звание подпись, дата фамилия, инициалы

Липецк 2023 г.

Оглавление

Задание кафедры

Реализовать алгоритм симметричного блочного шифрования по ГОСТ 28147-89 (режим простой замены, режим гаммирования, режим имитовставки). На контрольном примере проверить правильность работы алгоритмов шифрования и дешифрования.

1 Теоретическая часть

1.1 Алгоритм блочного шифрования по ГОСТ 28147-89. Режим простой замены

Шифрование:

1. Открытые данные, подлежащие зашифрованию, разбивают на 64- разрядные блоки T_0. Процедура зашифрования 64-разрядного блока T_0 в режиме простой замены включает 32 цикла. В КЗУ вводят 256 бит ключа в виде восьми 32-разрядных подключей (чисел) K_i:
   $K = K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7$.
2. Последовательность битов блока $T_0 = (a_1(0), ..., a_{32}(0), b_1(0), ..., b_{32}(0))$ разбивают на две последовательности по 32 бита: $b(0)$ — старшие биты, $a(0)$ — правые биты.
3. Начальное заполнение накопителя $N1$: $a(0) = (a_1(0), ..., a_{32}(0))$, накопителя $N2$ - $b(0)$.
4. Первый цикл зашифрования 64-разрядного блока открытых данных:

$$a(1) = f(a(0) + K_0) \oplus b(0)$$
$$b(1) = a(0)$$

5. Функция f включает две операции над полученной 32-разрядной суммой $(a(0) + K_0)$:
   — Подстановка (выполняется блоком подстановки S). Блок подстановки из восьми **S**-узлов замены с памятью 64 бит каждый. Поступающий из **CM1** на блок подстановки **S** 32-разрядный вектор разбивают на 8 последовательно идущих 4-разрядных векторов, каждый из которых преобразуется в четырехразрядный вектор соответствующим узлом замены. Каждый узел замены можно представить в виде таблицы-перестановки шестнадцати четырехразрядных двоичных чисел в диапазоне $0000...1111$. Входной вектор указывает адрес строки в таблице, а число в этой строке является выходным вектором. Затем четырехразрядные выходные векторы последовательно объединяют в 32-разрядный вектор.

— Циклический сдвиг влево (на 11 разрядов) 32-разрядного вектора, полученного с выхода блока подстановки **S**. Циклический сдвиг выполняется регистром сдвига **R**.

6. Результат работы функции шифрования f суммируют поразрядно по модулю 2 в сумматоре **СМ2** с 32-разрядным начальным заполнением $b(0)$ накопителя N_2.
7. Полученный на выходе **СМ2** результат (значение $a(1)$) записывают в накопитель N_1, а старое значение N_1 (значение a(0)) переписывают в накопитель N_2 (значение $b(1) = a(0)$). Первый цикл завершен.
   Порядок выборки подключей в 32 циклах:

$$K_0K_1K_2K_3K_4K_5K_6K_7K_0K_1K_2K_3K_4K_5K_6K_7$$
$$K_0K_1K_2K_3K_4K_5K_6K_7K_7K_6K_5K_4K_3K_2K_1K_0$$

8. В 32-м цикле результат из сумматора **СМ2** вводится в накопитель N_2, а в накопителе N_1 сохраняется прежнее заполнение. Полученные после 32-го цикла зашифрования заполнения накопителей N_1 и N_2 являются блоком зашифрованных данных T_o, соответствующим блоку T_0.
9. Порядок вывода блока зашифрованных данных:

$$T_o = (a_1(32), a_2(32), ..., a_{32}(32), b_1(32), b_2(32), ..., b_{32}(32)$$

Расшифрование:

1. В КЗУ вводят 256 бит ключа, на котором осуществлялось зашифрование. Зашифрованные данные, подлежащие расшифрованию, разбиты на блоки T_oпо 64 бита в каждом.
2. Порядок выборки подключей в 32 циклах расшифрования:

$$K_0K_1K_2K_3K_4K_5K_6K_7K_7K_6K_5K_4K_3K_2K_1K_0$$
$$K_7K_6K_5K_4K_3K_2K_1K_0K_7K_6K_5K_4K_3K_2K_1K_0$$

3. Полученные после 32 циклов работы заполнения накопителей N_1 и N_2 образуют блок открытых данных $T_0 = (a_1(0), ..., a_{32}(0), b_1(0), ..., b_{32}(0))$, соотвествующий блоку T_o.

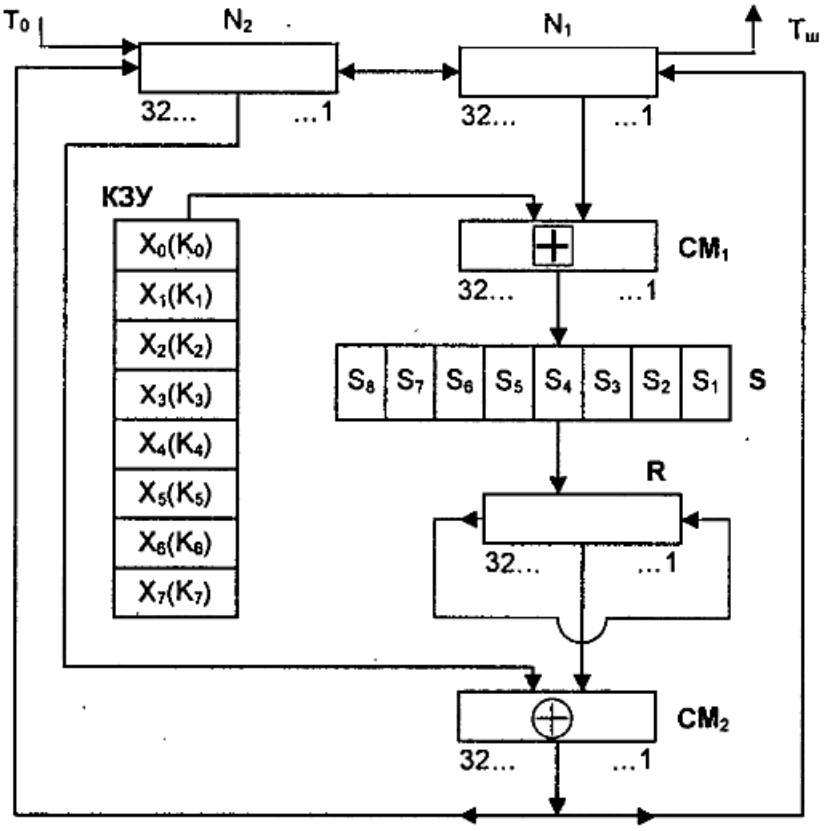

Рисунок 1 – Режим простой замены

2 Практическая часть

2.1 Реализация алгоритма шифрования блока

```
package block

type Gost struct {
    key []uint32
    s   [][]byte
}

type CipherGost interface {
    Encrypt(src []byte) []byte
    Decrypt(src []byte) []byte
}

func (g *Gost) Encrypt(src []byte) []byte {
    encSrc := bytesToUint32s(src)
    n := g.encrypt32(encSrc)
    resBytes := uint32sToBytes(n)
    return resBytes
}

func (g *Gost) Decrypt(src []byte) []byte {
    encSrc := bytesToUint32s(src)
    n := g.decrypt32(encSrc)
    resBytes := uint32sToBytes(n)
```

```
    return resBytes
}

func (g *Gost) f(x uint32) uint32 {
    x = uint32(g.s[0][(x>>24)&255])<<24 | uint32(g.s[1][(x>>16)&255])<<16 |
            uint32(g.s[2][(x>>8)&255])<<8 | uint32(g.s[3][x&255])
    return (x << 11) | (x >> (32 - 11))
}

func (g *Gost) encrypt32(src []uint32) []uint32 {
    var n = make([]uint32, 2)
    n[1], n[0] = src[0], src[1]

    for i := 0; i < 3; i++ {
            for j := 0; j < 8; j += 2 {
                    n[0] = n[0] ^ g.f(n[1]+g.key[j])
                    n[1] = n[1] ^ g.f(n[0]+g.key[j+1])
            }
    }

    for j := 7; j >= 0; j -= 2 {
            n[0] = n[0] ^ g.f(n[1]+g.key[j])
            n[1] = n[1] ^ g.f(n[0]+g.key[j-1])
    }

    return n
}

func (g *Gost) decrypt32(src []uint32) []uint32 {
    var n = make([]uint32, 2)
    n[1], n[0] = src[0], src[1]

    for j := 0; j < 8; j += 2 {
            n[0] = n[0] ^ g.f(n[1]+g.key[j])
            n[1] = n[1] ^ g.f(n[0]+g.key[j+1])
    }

    for i := 0; i < 3; i++ {
            for j := 7; j >= 0; j -= 2 {
                    n[0] = n[0] ^ g.f(n[1]+g.key[j])
                    n[1] = n[1] ^ g.f(n[0]+g.key[j-1])
            }
    }

    return n
}

func NewCipherGost() CipherGost {
    return &Gost{
            key: bytesToUint32s(TestKey),
            s:   STable(TestSBox),
    }

}

var TestKey = []uint8{0xbe, 0x5e, 0xc2, 0x00, 0x6c, 0xff, 0x9d, 0xcf,
    0x52, 0x35, 0x49, 0x59, 0xf1, 0xff, 0x0c, 0xbf,
    0xe9, 0x50, 0x61, 0xb5, 0xa6, 0x48, 0xc1, 0x03,
    0x87, 0x06, 0x9c, 0x25, 0x99, 0x7c, 0x06, 0x72}
```

```
var TestSBox = [][]byte{
    {4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3},
    {14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9},
    {5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11},
    {7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3},
    {6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2},
    {4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14},
    {13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12},
    {1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12},
}
```

2.2 Реализация алгоритма деления на блоки

```
package block

type Block []byte

type MessageBlocks struct {
    blocks []Block
    cipher CipherGost
}

type BlockEncoder interface {
    Encrypt() []byte
    Decrypt() []byte
    SetMessage([]byte)
}

func (g *MessageBlocks) SetMessage(message []byte) {
    var blocks = make([]Block, 0)
    for len(message)%8 != 0 {
            message = append(message, byte(32))
    }
    length := len(message)
    for i := 0; i < length; i += 8 {
            blocks = append(blocks, message[i:i+8])
    }
    g.blocks = blocks
}

func (g *MessageBlocks) Encrypt() []byte {
    result := make([]byte, 0)
    for i := range g.blocks {
            result = append(result, g.cipher.Encrypt(g.blocks[i])...)
    }
    return result
}

func (g *MessageBlocks) Decrypt() []byte {
    result := make([]byte, 0)
    for i := range g.blocks {
            result = append(result, g.cipher.Decrypt(g.blocks[i])...)
    }
    return result
}

func NewBlockEncoder(message []byte) BlockEncoder {
    g := &MessageBlocks{
            cipher: NewCipherGost(),
    }
```

```
    g.SetMessage(message)
    return g
}
```

2.3 Проверка программы

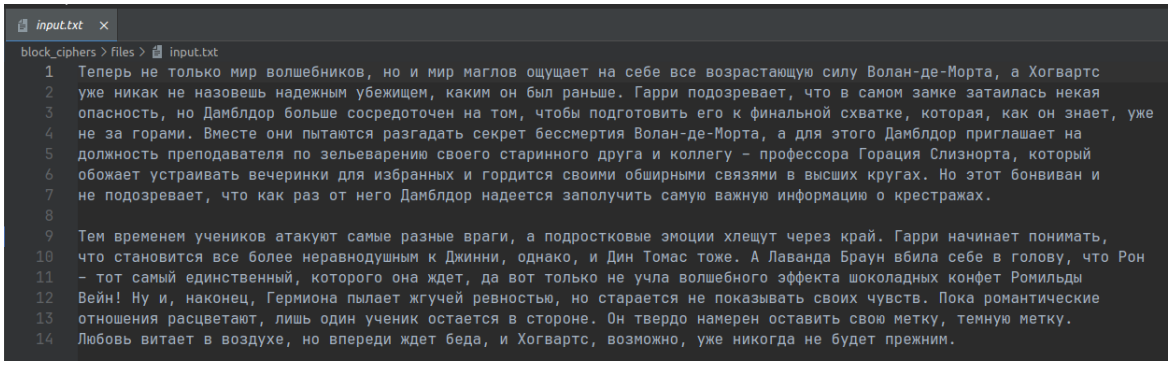

Рисунок 2 – Входные данные

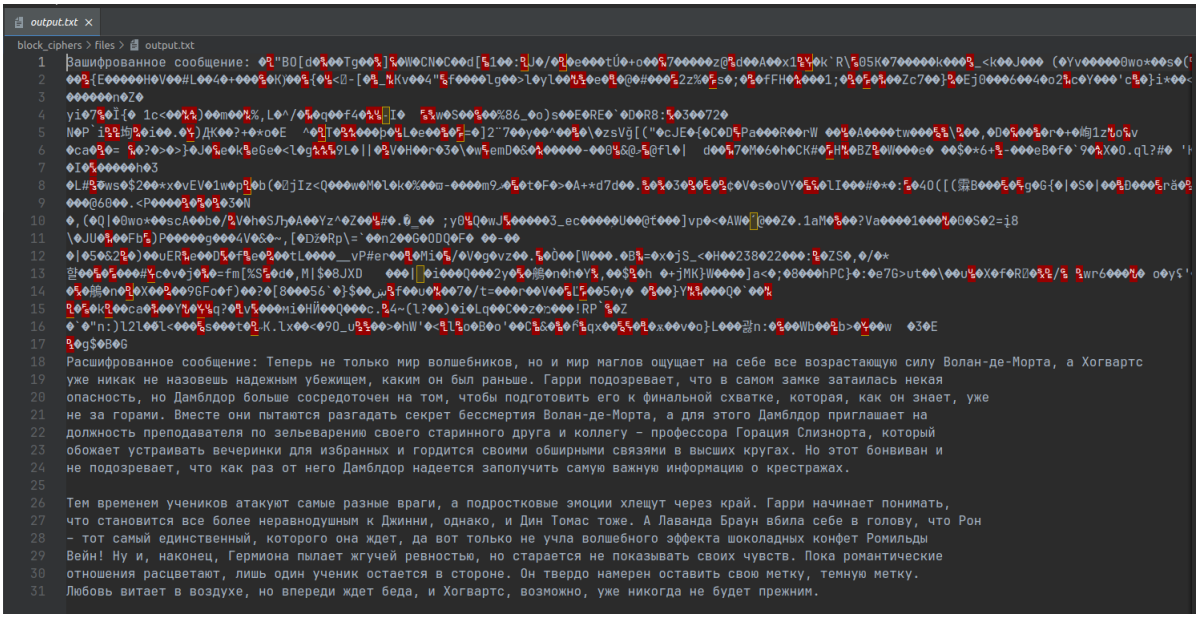

Рисунок 3 – Выходные данные

3 Контрольные вопросы

1. Принцип работы симметричных криптосистем.
   Симметричное шифрование — это способ шифрования данных, при котором один и тот же ключ используется и для кодирования, и для восстановления информации. До 1970-х годов, когда появились первые асимметричные шифры, оно было единственным криптографическим методом.
2. Структура ключа ГОСТ 28147-89.
   Размер ключа — 256 бит. Состоит из 8 32-разрядных блоков.
3. Синхропосылка ее значение.
   Синхропосылка — значения исходных открытых параметров алгоритма криптографического преобразования (64-разрядная двоичная последовательность).
4. Порядок использования ключевой информации.
   Порядок выборки подключей в 32 циклах:

$$K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7$$
$$K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7$$
$$K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7$$
$$K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0$$

   Порядок выборки подключей в 32 циклах расшифрования:

$$K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7$$
$$K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0$$
$$K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0$$
$$K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0$$

5. Сеть Фейстеля.
   Пусть требуется зашифровать некоторую информацию, представленную в двоичном виде (в виде последовательности нулей и единиц) и находящуюся в памяти компьютера или иного устройства (например, в файле).
   Алгоритм шифрования.

Информация разбивается на блоки одинаковой (фиксированной) длины. Полученные блоки называются входными, так как поступают на вход алгоритма. В случае если длина входного блока меньше, чем размер, который выбранный алгоритм шифрования способен зашифровать единовременно (размер блока), то блок удлиняется каким-либо способом. Как правило, длина блока является степенью двойки, например, составляет 64 бита или 128 бит.

Далее будем рассматривать операции, происходящие только с одним блоком, так как в процессе шифрования с другими блоками выполняются те же самые операции.

— Выбранный блок делится на два подблока одинакового размера — «левый» (L_0) и «правый» (R_0). «Правый подблок» R_0 изменяется функцией F с использованием раундового ключа K_0:
$x = F(R_0, K_0)$

— Результат складывается по модулю 2 («xor») с «левым подблоком» L_0:
$x = x \oplus L_0$

— Результат будет использован в следующем раунде в роли «правого подблока» R_1:
$R_1 = x$

— «Правый подблок» R_0 текущего раунда (в своем не измененном на момент начала раунда виде) будет использован в следующем раунде в роли «левого подблока» L_1:
$L_1 = R_0$

— «Правый подблок» R_0 текущего раунда (в своем не измененном на момент начала раунда виде) будет использован в следующем раунде в роли «левого подблока» L_1:
$L_1 = R_0$

Перечисленные операции выполняются $N - 1$ раз, где N — количество раундов в выбранном алгоритме шифрования. При этом между переходами от одного раунда (этапа) к другому изменяются ключи: K_0 заменяется на K_1, K_1 — на K_2 и т. д.).

Расшифрование.

Расшифровка информации происходит так же, как и шифрование, с тем лишь исключением, что ключи следуют в обратном порядке, то есть не от первого к N-му, а от N-го к первому.

6. Алгоритмы зашифрования, расшифрования и выработки имитовставки.

   Имитовставка - это блок из P бит, который вырабатывают по определенному правилу из открытых данных с использованием ключа и затем добавляют к зашифрованным данным для обеспечения их имитозащиты.

   — Имитовставка вырабатывается из блоков открытых данных либо перед шифрованием всего сообщения, либо параллельно с шифрованием по блокам. Первые блоки открытых данных, которые участвуют в выработке имитовставки, могут содержать служебную информацию (например, адресную часть, время, синхропосылку) и не зашифровываются.

   — Значение параметра (число двоичных разрядов в имитовставке) определяется криптографическими требованиями с учетом того, что вероятность навязывания ложных помех равна $\frac{1}{2}^P$.

   — Для выработки имитовставки открытые данные представляют в виде последовательности 64-разрядных блоков $T_0^{(i)}, i = 1, ..., m$.

   — Первый блок открытых данных $T_0^{(1)}$ подвергают преобразованию $\widetilde{A}()$, соответствующему первым 16 циклам алгоритма зашифрования в режиме простой замены. В качестве ключа для выработки имитовставки используют ключ длиной 256 бит, по которому шифруют данные.

   — Полученное после 16 циклов 64-разрядное число $\widetilde{A}(T_0^{(1)})$ суммируют по модулю 2 со вторым блоком открытых данных $T_0^{(2)}$. Результат суммирования $\widetilde{A}(T_0^{(1)}) \oplus T_0^{(2)}$ снова подвергают преобразованию $\widetilde{A}()$, и т.д...

   — Последний блок $T_0^{(m)}$ (при необходимости дополненный нулями до полного 64-разрядного блока) суммируют по модулю 2 с результатом вычислений на шаге $(m-1)$, после чего зашифровывают в режиме простой замены, используя преобразование $\widetilde{A}()$.

— Из полученного 64-разрядного числа выбирают отрезок (имитовставку) длиной P бит:

$$[a^{(m)}_{32-p+1}(16), a^{(m)}_{32-p+2}(16), ..., a^{(m)}_{32}(16)],$$

где $a^{(m)}_i$ — i-ый бит 64-разрядного числа, полученного после 16-ого цикла последнего преобразования $\widetilde{A}()$.

— Имитовставка передается по каналу связи или в память в конце зашифрованных данных.

— Поступившие к получателю зашифрованные данные расшифровываются, и из полученных блоков открытых данных аналогичным образом вырабатывается имитовставка. Эта имитовставка сравнивается с имитовставкой, полученной вместе с зашифрованными данными из канала связи или из памяти ЭВМ. В случае несовпадения имитовставок полученные при расшифровании блоки открытых данных считают ложными.

7. Отличия блочных и поточных симметричных шифров.
   Отличие поточных от блочных шифров состоит в способе шифрования, т.е. поточные — преобразуют текст посимвольно, а блочные работают с блоками информации, размер блока чаще всего составляет 64 или 128 битов.
8. Криптостойкость симметричных блочных шифров.
   Криптостойкость определяется:
   — Параметры ключа: его длина и длина блока, обеспечивающие верхнюю границу безопасности шифра.
9. Связь основного шага криптопреобразования и базовых циклов.
   а самом верхнем находятся практические алгоритмы, предназначенные для шифрования массивов данных и выработки для них имитовставки. Все они опираются на три алгоритма низшего уровня, называемые в тексте ГОСТа циклами . Эти фундаментальные алгоритмы упоминаются в данной статье как базовые циклы , чтобы отличать их от всех прочих циклов. Они имеют следующие названия и обозначения, последние приведены в скобках:
   — цикл зашифрования (32-З);
   — цикл расшифрования (32-Р);

— цикл выработки имитовставки (16-З).

В свою очередь, каждый из базовых циклов представляет собой многократное повторение одной единственной процедуры, называемой для определенности далее в настоящей работе основным шагом криптопреобразования. Основной шаг криптопреобразования по своей сути является оператором, определяющим преобразование 64-битового блока данных.

10. Разбиение информации на блоки.

    Сообщение делится на блоки по 64 бита (8 байтов).

Заключение

В ходе выполнения данной работы был реализован алгоритм симметричного блочного шифрования по ГОСТ 28147-89 с режимом простой замены. На контрольном примере были проверены правильность и корректность работы приведённых алгоритмов шифрования и дешифрования.