Лабораторная работа №4

Блочные шифры, режимы использования блочных шифров

Цель работы: Научиться реализовывать простейшие блочные алгоритмы шифрования и режимы их использования и строить на их основе более сложные блочные шифры

Теоретическая часть

Блочный шифр использует итерационную модель последовательной обработки блоков постоянного размера. При шифровании каждого отдельного блока используется функция шифрования, которая обычно состоит из более простых шифрующих преобразований.

Основные виды шифрующих преобразований:

- перестановка (permutation) – перестановка структурных элементов шифруемого блока данных (битов, символов, цифр);
- замена, подстановка (substitution) – замена группы элементов шифруемого блока на другую группу по индексной таблице;
- функциональное преобразование (function) – различные сдвиги, логические и арифметические операций.

Примеры простых шифрующих преобразований

1. Шифр перестановки

В алгоритме *перестановки* в каждом блоке меняется последовательность некоторых подблоков внутри блока, например байт или бит в слове, причем порядок перестановок определяется ключом.

Пусть имеется некоторое исходное сообщение «`M E S S A G E `» , которое необходимо закодировать (Рис. 1). Это сообщение имеет длину в 7 байт (если используется ASCII код). Разобьем этот блок текста на три подблока: «`M E S `», «`S A `» и «`G E `». Числа *M* и *N*, которые определяют

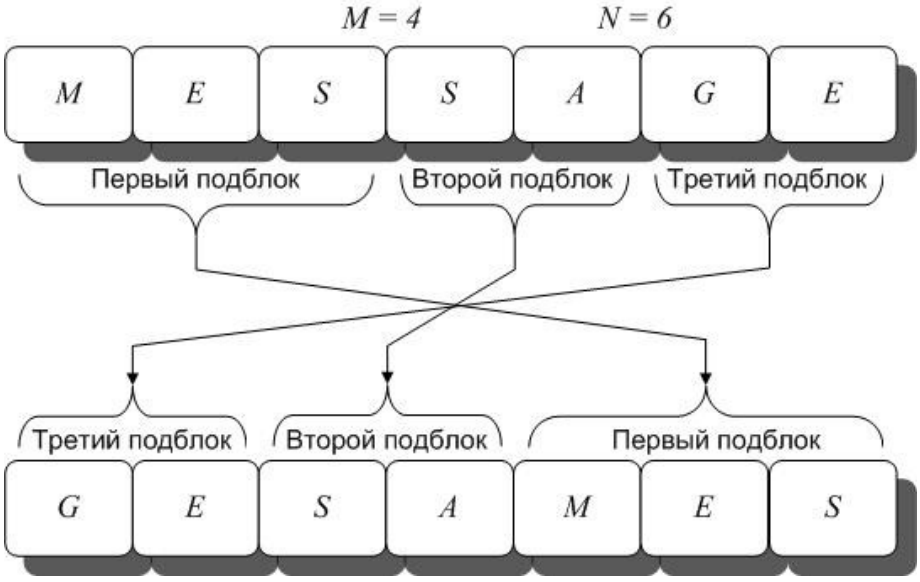


Рисунок 1. Пример перестановки

границы подблоков, получены при помощи генератора псевдослучайных чисел и зависят от конкретного ключа.

Для данного примера *M* = 4, *N* = 6.

После перестановки мы получим зашифрованное сообщение: «G E S A M E S » .

Дешифрование происходит по той же схеме, но в обратном порядке.

Механизм шифрования/дешифрования этого блочного шифра реализуется при помощи следующего алгоритма:

```
// Функция производящая перестановку в блоке
// str - исходный текст до перестановки
// n и m - границы подблоков в блоке
// return - текст после перестановки
// Правило формирования:
// 1. блок разбивается на три подблока, границами которых служат числа m и n
// 2. Третий подблок записывается на место первого
// 3. Второй  - на место второго
// 4. Первый - на место третьего
function TForm1.cryptblock (str:block;n,m:integer):block;
var
  // Счетчики внутренних и внешнего циклов
  i,k:integer;
  // Текст после выполнения перестановки
  retstr :block;
begin
  k := 1;
  // Третий подблок записывается на место первого
  for i:=n to sblock do begin
    retstr[k] := str[i];
    k:=k+1;
  end;
  // Второй  - на место второго
  for i:=m to n-1 do begin
    retstr[k] := str[i];
    k:=k+1;
  end;
  // Первый - на место третьего
  for i:=1 to m-1 do begin
    retstr[k] := str[i];
    k:=k+1;
  end;
  cryptblock := retstr;
end;
```

Аналогично производится операция перестановки над группами бит.

Примечание: для блоков большой длины метод перестановок становится неэффективным, так как и длина подблоков в этом случае также велика. Поэтому шифрование должно осуществляться в несколько раундов, то есть перемешивание производится несколько раз над одним и тем же блоком, но с различными значениями коэффициентов M и N. При дешифровании необходимо воспроизвести эту последовательность псевдослучайных чисел в обратном порядке.

2. Скремблирование (метод сдвига)

В алгоритме шифрования *методом сдвига (скремблера)* исходный текст разбивается на подблоки и внутри каждого такого подблока реализуется операция циклического сдвига на несколько бит в указанном направлении. Например, для блока длиной в 7 байт, где шифрование осуществляется циклическим сдвигом влево на 3 бита, схема шифрования будет следующей (Рис. 2):

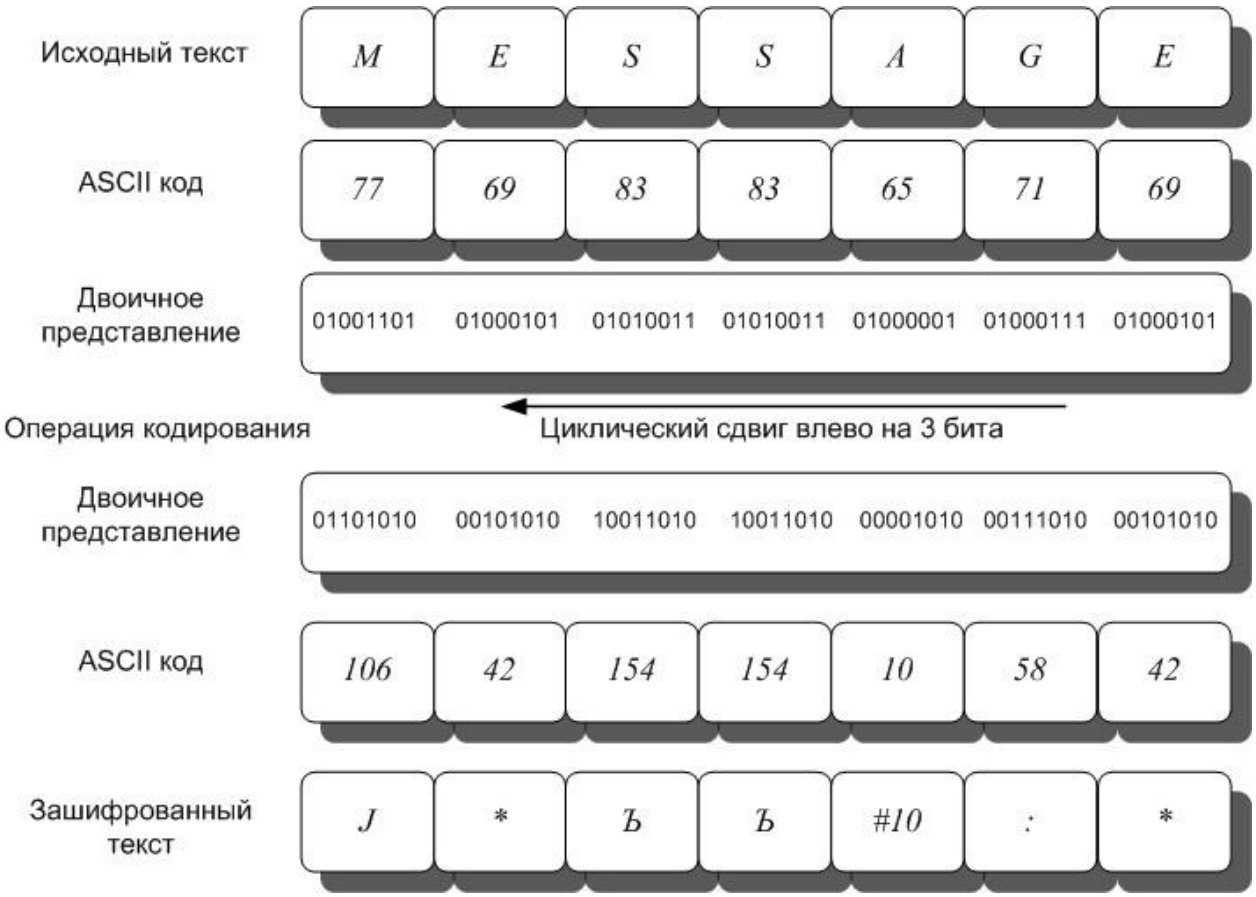


Рисунок 2. Пример шифрования методом циклического сдвига

Соответственно для раскодирования необходимо произвести циклический сдвиг подблока в противоположную сторону на такое же количество бит. Обычно величина сдвига и его направление определяется паролем, что обеспечивает привязку к ключу. В том случае, если блок шифруется по байтам, то следует исключать ситуации, при которых размер сдвига кратен 8.

Программная реализация такого шифра включает три функциональных блока, где каждый может быть оформлен как отдельная процедура.

- Процедура «распаковки» в битовое представление - unpack
- Процедура, организующая циклический сдвиг - shiftblock,
- Процедура «упаковки» битового представления обратно в текстовый блок - pack

```
// Функция преобразования байтового блока в массив бит
// cblock - текущий кодируемый блок
// cbitblock - битовое представление cblock
// Правило преобразования: все байты исходного блока переводятся в
// битовое представление (8 -> 2) и поочередно записываются в массив бит
// Например, если исходный блок - "AB" (#65#66), то массив бит будет таким: 01000001 1000010

procedure TForm1.unpack(cblock:block; var cbitblock:bitblock);
var
  // Счетчики циклов
  i,j:integer;
begin
  // Перебор байтов в блоке
  for i:=1 to sblock do begin
    // Перевод в 2-ю систему счисления
    for j:=8 downto 1 do begin
      // берем остаток от деления на 2 от текущего байта
      cbitblock[(i-1)*8+j]:=ord(cblock[i]) mod 2;
      // производим целочисленное деление текущего байта на 2
      cblock[i]:=chr(ord(cblock[i]) div 2);
    end;
```

```
  end;
end;

// Функция преобразования массива бит в байтовый блок
// cblock - текущий кодируемый блок
// cbitblock - битовое представление cblock
// Правило преобразования: битовое представление массива свертывается
// в байтовый блок, то есть производится действие обратное функции pack

procedure TForm1.pack(var cblock:block; cbitblock:bitblock);
var
  // Счетчики циклов
  i,j:integer;
begin
  // Поочередно формируем байты блока
  for i:=1 to sblock do begin
    cblock[i] := #0;
    for j:=1 to 8 do
      // "Свертываем" битовое представление в байт
      cblock[i] := chr((ord(cblock[i]) shl 1) or cbitblock[(i-1)*8+j]);
  end;
end;

// Функция, производящая сдвиг массива бит на n разрядов в указанном направлении
// cbitblock - исходный массив бит
// n - размер сдвига
// shift - направление сдвига: sright - сдвиг вправо, sleft  - сдвиг влево
// return - массив бит, сдвинутый на n разрядов в указанном направлении
// Правило преобразования: массив бит делится на две части, где n - граница
// и происходит перестановка полученных подблоков местами

function TForm1.shiftblock(cbitblock:bitblock; n:integer; shift:tshift):bitblock;
var
  // Счетчики внутреннего и внешнего циклов
  i,k:integer;
  // Массив для хранения текущего получаемого значения
  cshiftblock:bitblock;
begin
  k:=1;
  // сдвиг вправо
  if shift = sright then begin
    for i:=sblock*8-n+1 to sblock*8 do begin
      cshiftblock[k]:= cbitblock[i];
      k:=k+1;
    end;
    for i:=1 to sblock*8-n do begin
      cshiftblock[k]:= cbitblock[i];
      k:=k+1;
    end;
  end else
  // сдвиг влево
  begin
    for i:=n+1 to sblock*8 do begin
      cshiftblock[k]:= cbitblock[i];
      k:=k+1;
    end;
    for i:=1 to n do begin
      cshiftblock[k]:= cbitblock[i];
      k:=k+1;
    end;
  end;
  shiftblock := cshiftblock;
end;
```

3. Замена по таблице

Метод замены по таблице, использует специальную функцию «потопления статистики», которая значительно затрудняет взлом. В табличном методе при шифровании одному символу может быть поставлено в соответствие одно из нескольких значений, которое выбирается абсолютно случайно и не зависит от ключа. Это означает, что даже используя один и тот же ключ для шифрования одинакового текста, каждый раз мы будем получать разные шифры.

Для этого необходимо сформировать таблицу значений для подстановки. Размерность этой таблицы зависит от двух параметров алгоритма: длины шифруемого блока (n) и некоторой

постоянной величины, которая определяет число возможных значений подстановки для шифрования одного и того же символа. Обозначим эти параметры как rows – число строк в таблице и columns – число столбцов. Число строк определяется как rows = 2n, где n – это длина шифруемого блока в битах. Полученная таблица размерностью заполняется псевдослучайными значениями от 0 до rows*columns. При этом обязательным требованием должно быть требование уникальности этих значений, то есть они не должны повторяться.

Этого можно добиться следующим образом:

```
// Процедура генерации таблицы значений для подстановки
// tab - таблица для подстановочных значений
// Правило формирования:
// Шаг 1: Таблица заполняется последовательными значениями от 0 до columns*rows
// Шаг 2: Значения в таблице перемешиваются counthash раз в зависимости от значений, выдаваемых
// собственным конгруэнтным генератором, который инициализируется hash значением

procedure TForm1.genTable(var tab:ttab);
var
      // Внутренний и внешний счетчики циклов
  i,j:integer;
     // Массив для хранения 2-х координат таблицы.
  // Значения с такими координатами меняются местами
  rr:trr;
     // Предыдущее значение выданное конгруэнтным генератором
  prev:cardinal;
      // Буфер для организации обмена двумя числами в таблице
  temp:integer;
begin
  // Начальное заполнение таблицы последовательными значениями
  for i:=0 to rows-1 do
    for j:=0 to columns-1 do
      tab[i,j] := i*columns+j;
  // Извлекаем первое случайное значение из конгруэнтного генератора
  prev := genCongr(hash(PasswordDlg.Password.Text));
  // Организуем цикл перемешивания таблицы в counthash шагов
  for j := 0 to counthash do begin
    // Заполняем массив rr четырьмя случайными значениями
    for i:=0 to 3 do
      if i mod 2 =0 then begin
        prev:=genCongr(prev);
        rr[i] := prev mod rows;
      end
      else begin
        prev := genCongr(prev);
        rr[i]:=prev mod columns;
      end;
    // Меняем элементы местами в зависимости от полученных координат
    temp := tab[rr[0]][rr[1]];
    tab[rr[0]][rr[1]] := tab[rr[2]][rr[3]];
    tab[rr[2]][rr[3]] := temp;
  end;
end;
```

В приведенном алгоритме используется прием перемешивания значений таблицы в зависимости от пароля. То есть изначально таблица заполняется последовательными значениями, а затем происходит перетасовка элементов. В результате получается таблица, заполненная псевдослучайными значениями, которая и называется таблицей подстановки.

Допустим, что размер группы *4* бита, а число столбцов равно *3*. Значит, таблица будет иметь размерность $2^4 \times 3 = 16 \times 3$ и должна заполняться значениями в диапазоне от *0* до *48*.

Например, в нашем случае может получиться такая таблица:

	1	2	3
0	15	10	2
1	3	17	9
2	8	14	1
3	0	16	45
4	…	…	…
7	20	5	38
8	…	…	…
15	12	30	4

При шифровании исходный блок представляется в виде числового эквивалента и в таблице отыскивается строка, номер которой равен этому эквиваленту. Далее в этой строке абсолютно случайно выбирается одно из нескольких значений.

Например, был прочитан блок, числовой эквивалент которого равен *7*. По таблице в строке с номером *7* находится тройка значений (*20, 5, 38*). При помощи функции *random(3)* выбирается любое значений из этой тройки. Это значение и есть зашифрованный блок.

На языке Object Pascal такой алгоритм можно представить следующим образом:

```
// Функция шифрования
// cblock - текущий блок, подлежащий шифрованию
// return - зашифрованный блок двойного размера
// Правило формирования:
// Шаг 1: Текущий кодируемый блок переводится в число
// Шаг 2: Производится замена по таблице
// Шаг 3: Значение из таблицы переводится в строковое представление

function TForm1.encryptblock(cblock:block):dblock;
var

  value:integer;    // Текущий числовой эквивалент кодируемого значения
i retblock: dblock; // Возвращаемый зашифрованный блок
begin
  value := 0;
  // Переводим кодируемый блок в числовой эквивалент
  for i:=1 to sblock do
    value := (value shl 8) or ord(cblock[i]);
  // Берем значение из таблицы
  value := tab[value][random(columns)];
  // Переводим полученное числовое значение в текстовый эквивалент
  for i:=1 to sblock*2 do begin
    // Считаем циклы
    retblock[i] := chr(value and $00ff);
    value := value shr 8;
  end;
  encryptblock := retblock;
end;
```

При дешифровании сначала воспроизводится таблица подстановки, так как она зависит от ключа. Затем читается закодированный блок и по таблице отыскивается строка, в которой присутствует это значение. Например, если был прочитан зашифрованный блок, числовой эквивалент которого равен *16*, то в таблице такой значение находится в строке с номером *3*. Это и есть требуемое значение. Номер строки преобразуется в строковое представление и записывается в выходной поток.

Ниже приведен соответствующий алгоритм:

```
// Функция дешифрования
// cblock - текущий блок двойного размера, подлежащий дешифрованию
// return - расшифрованный блок одинарного размера
// Шаг 1: Текущий декодируемый блок переводится в число
// Шаг 2: Производится поиск значения в таблице
// Шаг 3: Номер строки таблицы переводится в строковое представление

function TForm1.decryptblock(cblock:dblock):block;
var
  i,j,        // Счетчики внешнего и внутреннего циклов
  value,      // Текущий числовой эквивалент декодируемого блока
  temp:integer;   // Временная переменная
  retblock:block; // Возвращаемый декодирвоанный блок
begin
  value := 0;
  // Переводим блок в число
  for i:=sblock*2 downto 1 do
    value := (value shl 8) or ord(cblock[i]);
  j:=0;
  i:=0;
  // Ищем значение в таблице
  while temp <> value do begin
    if j div columns-1 = 0 then begin
      i:=i+1;
      j:=0;
    end;
    temp := tab[i][j];
    j := j+1;
  end;
  // Переводим полученное значение в стоковое представление
  for j:=1 to sblock do begin
    retblock[sblock-j+1]:=chr(i and $ff);
    i := i shr 8;
  end;
  decryptblock := retblock;
end;
```

Как было сказано раньше, такой алгоритм обладает высокой криптостойкостью за счет абсолютной случайности выбора при шифровании. Однако он имеет существенный недостаток – это увеличение объема шифруемого текста, что не является хорошим показателем.

4. Матричное шифрование

Для шифрования информации могут использоваться матричные преобразования. Наибольшее распространение получили методы шифрования, основанные на использовании матричной алгебры. Зашифрование k-го блока исходной информации, представленного в виде вектора $B_k = |b_j|$, осуществляется путем перемножения матрицы-ключа $A = |a_{ij}|$ и вектора B_k. В результате перемножения получается блок шифртекста в виде вектора $C_k = |c_i|$, где элементы вектора C_k определяются по формуле:

$$C_i = \sum_j a_{ij} b_j$$

Расшифрование информации осуществляется путем последовательного перемножения векторов C_k и матрицы A^{-1}, обратной матрице A.

Пример шифрования информации с использованием алгебры матриц.

Пусть необходимо зашифровать и расшифровать слово

Т = < ЗАБАВА> с помощью матрицы-ключа А (ключ матрицы задается для каждого блока с использованием генератора псевдослучайных чисел):

$$A = \begin{vmatrix} 1 & 4 & 8 \\ 3 & 7 & 2 \\ 6 & 9 & 5 \end{vmatrix}$$

Для шифрования исходного слова необходимо выполнить следующие шаги.

Шаг 1. Определяется числовой эквивалент исходного слова как последовательность соответствующих порядковых номеров букв слов Тэ:

Тэ = <8, 1, 2, 1, 3, 1>

Шаг 2. Умножение матрицы А на векторы В1 = {8, 1, 2} и В2 = {1, 3, 1}:

$$C_2 = \begin{vmatrix} 1 & 4 & 8 \\ 3 & 7 & 2 \\ 6 & 9 & 5 \end{vmatrix} \bullet \begin{vmatrix} 1 \\ 3 \\ 1 \end{vmatrix} = \begin{vmatrix} 21 \\ 26 \\ 38 \end{vmatrix}$$

Шаг 3. Зашифрованное слово записывается в виде последовательности чисел

Тш = <28, 35, 67, 21, 26, 38>.

Дешифрование слова осуществляется следующим образом.

Шаг 1. Вычисляется определитель |A| = -115.

Шаг 2. Определяется присоединенная матрица А*, каждый элемент которой является алгебраическим дополнением элемента матрицы А

$$A^* = \begin{vmatrix} 17 & -3 & -15 \\ 52 & -43 & 15 \\ -48 & 22 & -5 \end{vmatrix}$$

Шаг 3. Получается транспонированная матрица A^T

$$A^T = \begin{vmatrix} 17 & 52 & -48 \\ -3 & -43 & 22 \\ -15 & 15 & -5 \end{vmatrix}$$

Шаг 4. Вычисляется обратная матрица A^{-1} по формуле: $A^{-1} = AT/|A|$.

В результате вычислений обратная матрица имеет вид:

$$A^{-1} = \begin{vmatrix} -17/115 & -52/115 & 48/115 \\ 3/115 & 43/115 & -22/115 \\ 15/115 & -15/115 & 5/115 \end{vmatrix}$$

Шаг 5. Определяются векторы В1 и В2:

$$B_1 = \begin{vmatrix} -17/1151 & -52/115 & 48/115 \\ 3/115 & 43/115 & -22/115 \\ 15/115 & -15/115 & 5/115 \end{vmatrix} \bullet \begin{vmatrix} 28 \\ 35 \\ 67 \end{vmatrix} = \begin{vmatrix} 8 \\ 1 \\ 2 \end{vmatrix} \quad B_1 = \begin{vmatrix} -17/115 & -52/115 & 48/115 \\ 3/115 & 43/115 & -22/115 \\ 15/115 & -15/115 & 5/115 \end{vmatrix} \bullet \begin{vmatrix} 21 \\ 26 \\ 38 \end{vmatrix} = \begin{vmatrix} 1 \\ 3 \\ 1 \end{vmatrix}$$

Шаг 6. Числовой эквивалент расшифрованного слова

Тэ = <8, 1, 2, 1, 3, 1> заменяется символами, в результате чего получается исходное слово

Т = <ЗАБАВА>

Режимы использования блочных шифров

Для шифрования исходного текста произвольной длины блочные шифры могут быть использованы в нескольких режимах. Чаще всего используются четыре основных режима. Это режим *электронной кодировочной книги* (ECB – Electronic Code Book), *сцепления блоков шифрованного текста* (CBC – Cipher Block Chaining), *обратной связи по шифрованному тексту*

1. Режим ECB

В режиме *электронной кодировочной книги (ECB)* каждый блок исходного текста шифруется блочным шифром независимо от других (Рис. 3).

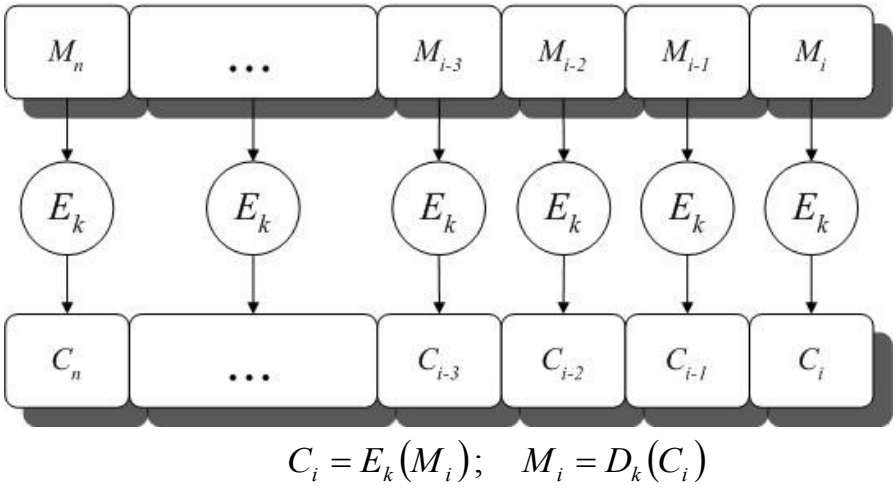


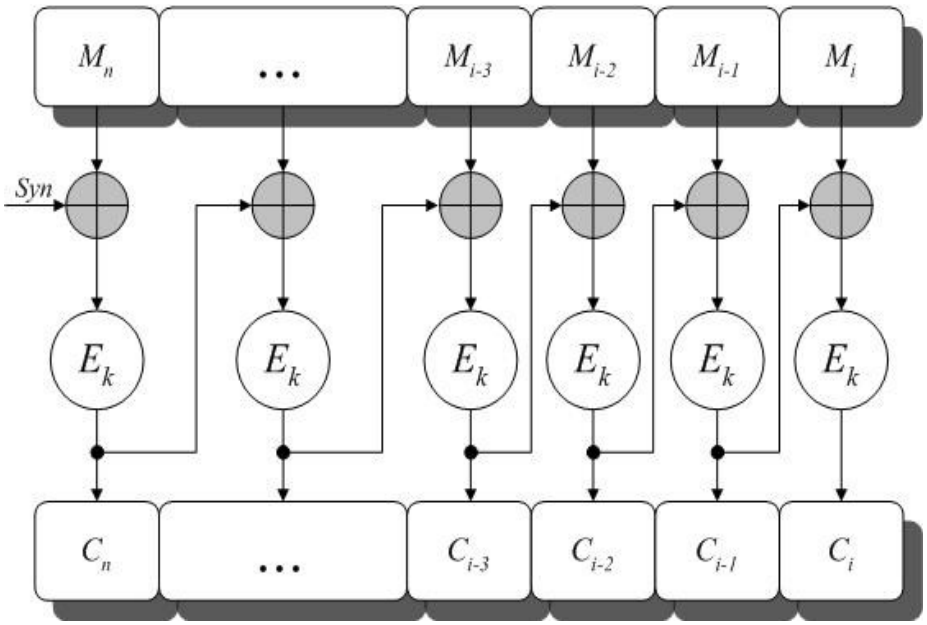
Рисунок 3. Режим электронной кодировочной книги (ECB)

Стойкость режима ECB равна стойкости самого шифра, однако структура шифрованного текста при этом не скрывается, так как каждый одинаковый блок исходного текста приводит к появлению одинакового блока шифрованного текста. Режим ECB допускает простое распараллеливание для увеличения скорости шифрования. Режим ECB соответствует режиму простой замены алгоритма ГОСТ 28147-89.

2. Режим CBC

В режиме *сцепления блоков шифрованного текста (CBC)* каждый блок исходного текста складывается поразрядно по модулю два с предыдущим блоком шифрованного текста, а затем шифруется (Рис. 4). Для начала процесса шифрования используется *синхропосылка* (или начальный вектор), которая передается в канал связи в открытом виде.

Стойкость режима CBC равна стойкости блочного шифра, лежащего в его основе. Кроме того, структура исходного текста скрывается за счет сложения предыдущего блока шифрованного текста с очередным блоком открытого текста. Стойкость шифрованного текста увеличивается, поскольку становится невозможной простая манипуляция исходным текстом, кроме как путем удаления блоков из начала или конца шифрованного текста. Скорость шифрования равна скорости работы блочного шифра, но простого способа распараллеливания процесса шифрования не существует, хотя дешифрование может проводиться параллельно.

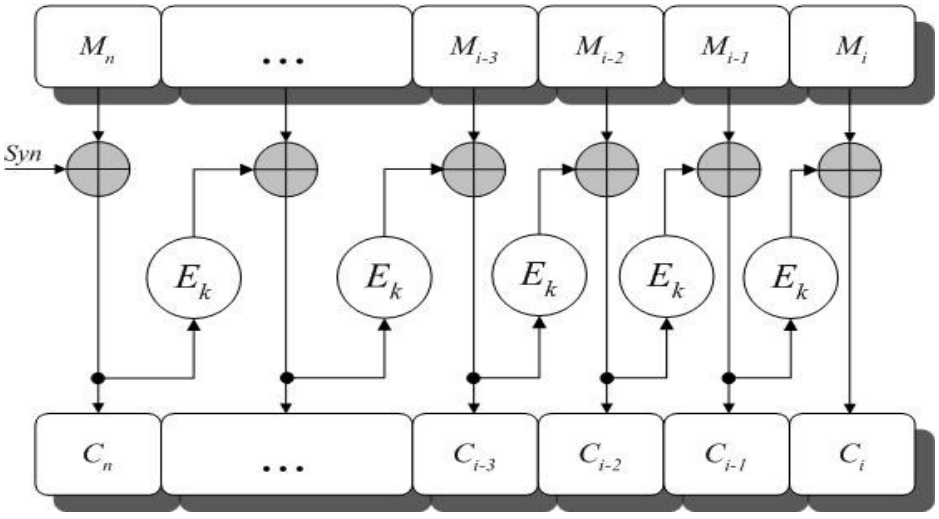


$$C_i = E_k\left(M_i \oplus C_{i-1}\right);\ M_i = D_k\left(C_i\right) \oplus C_{i-1}$$

Рисунок 4. Режим сцепления блоков шифрованного текста (CBC)

3. Режим CFB

В режиме *CFB* каждый блок исходного текста шифруется, а затем складывается поразрядно по модулю два с предыдущим блоком шифрованного текста (Рис. 5). Для начала процесса шифрования используется *синхропосылка* (или начальный вектор), которая передается в канал связи в открытом виде.



$$C_i = M_i \oplus E_k\left(C_{i-1}\right);\quad M_i = C_i \oplus D_k\left(C_{i-1}\right)$$

Рисунок 5. Режим обратной связи по шифрованному тексту (CFB)

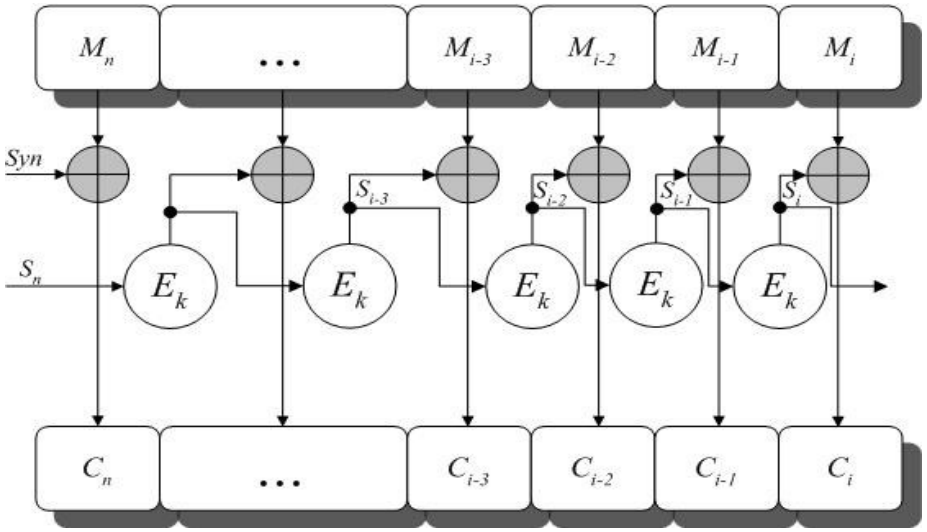
Стойкость режима CFB равна стойкости блочного шифра, лежащего в основе и структура исходного текста скрывается за счет использования операции сложения по модулю 2. Манипулирование исходным текстом путем удаления блоков из начала или конца шифрованного

текста становится невозможным. Однако в режиме CFB при шифровании двух идентичных блоков на следующем шаге шифрования также получатся идентичные зашифрованные блоки, что создает возможность утечки информации об исходном тексте.

Скорость шифрования равна скорости работы блочного шифра и простого способа распараллеливания процесса шифрования также не существует. Этот режим в точности соответствует режиму гаммирования с обратной связью алгоритма ГОСТ 28147–89.

4. Режим OFB

Режим *обратной связи по выходу (OFB)* подобен режиму CFB за исключением того, что величины, складываемые по модулю 2 с блоками исходного текста, генерируются независимо от исходного или шифрованного текста (Рис. 6). Для начала процесса шифрования также используется начальный вектор. Режим OFB обладает преимуществом перед режимом CFB в том смысле, что



$$C_i = M_i \oplus S_i \;; \quad M_i = C_i \oplus S_i \;; \quad S_i = E_k\left(S_{i-1}\right)$$

Рисунок 6. Режим обратной связи по выходу (OFB)

любые битовые ошибки, возникшие в процессе передачи, не влияют на дешифрование последующих блоков. Однако возможна простая манипуляция исходным текстом путем изменения шифрованного текста.

Блочные алгоритмы шифрования

1. Алгоритм DES

Data Encryption Standard — симметричный алгоритм шифрования, в котором один ключ используется как для шифрования, так и для расшифрования данных. DES разработан фирмой IBM и утвержден правительством США в 1977 году как официальный стандарт (FIPS 46-3). DES имеет блоки по 64 бит и 16 цикловую структуру сети Фейстеля, для шифрования использует ключ с длиной 56 бит(в преобразовании учавствуют раундовые ключи по 48 бит). Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований.

Описание доступно по ссылке: https://ru.bmstu.wiki/DES_(Data_Encryption_Standard)

2. Алгоритм AES

Американский стандарт, опубликованный в 2001 году. В современный криптографических подуктах, наверно, не найдется таких, которые бы не использовали AES. Используется в WI-FI, WinRAR, PGP. DES- предшественник AES. AES - симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES.

Описание доступно по ссылке: https://ru.bmstu.wiki/AES_(Advanced_Encryption_Standard)

3. Алгоритм ГОСТ

ГОСТ 28147-89 — советский и российский стандарт Симметричное шифрование симметричного шифрования, введённый в 1990 году, также является стандартом СНГ. Полное название— «ГОСТ 28147-89 Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования». Блочный шифроалгоритм. При использовании метода шифрования с гаммированием, может выполнять функции поточного шифроалгоритма.

Описание доступно по ссылке: https://ru.bmstu.wiki/ГОСТ_28147-89

Задание

Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия:

1. Шифровать и дешифровать текстовые и двоичные файлы с помощью блочного шифра
2. Сохранять зашифрованные/дешифрованные данные в файл
3. Загружать зашифрованные/дешифрованные данные из файла
4. Хешировать текстовый пароль, который используется при шифровании для инициализации генератора псевдослучайных чисел, используемого в блочных шифрах, с помощью функции хеширования, реализованной в предыдущей лабораторной работе.

Дополнительные требования к приложению

- Программа должна быть оформлена в виде удобной утилиты
- Программа должна обрабатывать файлы любого размера и содержания
- Текст программы оформляется прилично (удобочитаемо, с описанием ВСЕХ функций, переменных и критических мест).
- В процессе работы программа ОБЯЗАТЕЛЬНО выдает информацию о состоянии процесса шифрования / дешифрования.
- Интерфейс программы может быть произвольным, но удобным и понятным (разрешается использование библиотек GUI)
- Среда разработки и язык программирования могут быть произвольными.

Примечание: Задание является дифференцированным по сложности. Варианты заданий с указанием сложности приведены в таблице «Варианты задания»

Требования для сдачи лабораторной работы:

- Демонстрация работы реализованной вами системы.
- АВТОРСТВО
- Теория (ориентирование по алгоритмам и теоретическим аспектам методов тестирования)
- Оформление и представление письменного отчета по лабораторной работе, который содержит:
  - Титульный лист
  - Задание на лабораторную работу
  - Описание используемых алгоритмов шифрования
  - Листинг программы

Варианты задания

№	На оценку «Удовлетворительно» (в скобках указан размер блока)	На оценку «Хорошо»	На оценку «Отлично»
1.	Алгоритм перестановки (5 байт)	+ режим CBC	DES (режим CBC)
2.	Скремблирование (5 байт)	+ режим CFB	AES (блок и ключ 128 бит, 10 раундов)
3.	Замена по таблице (1 байт)	+ режим OFB	ГОСТ (режим простой замены)
4.	Матричное шифрование (5 байт)	+ режим CBC	DES (режим CFB)
5.	Алгоритм перестановки (6 байт)	+ режим CFB	AES (блок и ключ 192 бита, 10 раундов)
6.	Скремблирование (6 байт)	+ режим OFB	ГОСТ (Режим гаммирования)
7.	Замена по таблице (2 байта)	+ режим CBC	DES (режим OFB)
8.	Матричное шифрование (6 байт)	+ режим CFB	AES (блок и ключ 256 бит, 10 раундов)
9.	Алгоритм перестановки (7 байт)	+ режим OFB	ГОСТ (гаммирование с обратной связью)
10.	Скремблирование (7 байт)	+ режим CBC	DES (режим ECB +схема DES-EEE3)
11.	Замена по таблице (3 байта)	+ режим CFB	AES (блок и ключ 128 бит, 12 раундов)
12.	Матричное шифрование (7 байт)	+ режим OFB	ГОСТ (режим простой замены)
13.	Алгоритм перестановки (8 байт)	+ режим CBC	DES (режим ECB +схема DES-EDE3)
14.	Скремблирование (8 байт)	+ режим CFB	AES (блок и ключ 192 бита, 12 раундов)
15.	Матричное шифрование (8 байт)	+ режим OFB	ГОСТ (Режим гаммирования)