МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибернетики
Кафедра информационной безопасности

КУРСОВАЯ РАБОТА
по дисциплине
«Методы программирования»

Тема курсовой работы
«Реализация шифратора на основе алгоритма ГОСТ Р 34.12-2018 - «Кузнечик» в режиме простой замены с зацеплением»

Студент группы ККСО-03-19: Даваев В.П. _____

Руководитель курсовой работы: Кирюхин В.А. _____

Работа представлена к защите «____» _____ 2021 г.

Допущен к защите «____» _____ 2021 г.

Москва 2021

Содержание

Введение

Еще совсем недавно такая наука как криптография была доступна специальным службам безопасности и применялась для защиты засекреченных данных. Однако благодаря прорыву, произошедшему за последние десятилетия, появилась потребность в усиленной защите любой сферы деятельности, которая активно использует информационные системы и технологии.Эта необходимость позволила преподавать и вообще говорить о криптографических методах защиты данных,поэтому теперь даже обычные студенты могут получить пусть и простейшие(первичные), но очень важные практические уроки по данной дисциплине.

Одним из современных решений защиты информации является шифр ГОСТ Р 34.12-2018 «Кузнечик», который будет описан в этой работе.

Целью курсовой работы является изучение и реализация современного шифра ГОСТ Р 34.12-2018 «Кузнечик» и режима простой замены с зацеплением.

1. Теоритическая часть

1.1. ГОСТ 34.12-2018 «Кузнечик»

Данный шифр является симметричным алгоритмом блочного шифрования с размером блока 128 бит(16 байт) и длиной ключа 256 бит(32 байта)

Состоит из следующих преобразований:

$$X[k] : V_{128} \to V_{128} : X[k](a) = k \oplus a$$

Здесь строка a, длиной 128 бит(16 байт), ксорится со строкой k длиной 128 бит(16 байт)

$$S : V_{128} \to V_{128} S(a_{15}||...a_0) = \pi(a_{15})||...||\pi(a_0), a_i \in V_8, i = 0, 1, ..., 15$$

π = (252, 238, 221, 17, 207, 110, 49, 22, 251, 196, 250, 218, 35, 197, 4, 77, 233, 119, 240, 219, 147, 46, 153, 186, 23, 54, 241, 187, 20, 205, 95, 193, 249, 24, 101, 90, 226, 92, 239, 33, 129, 28, 60, 66, 139, 1, 142, 79, 5, 132, 2, 174, 227, 106, 143, 160, 6, 11,237, 152, 127, 212, 211,31,235, 52, 44, 81, 234, 200, 72, 171,242, 42, 104, 162, 253, 58, 206, 204, 181, 112, 14, 86, 8, 12, 118, 18, 191, 114, 19, 71, 156, 183, 93, 135, 21,161, 150, 41, 16, 123, 154, 199, 243, 145, 120, 111, 157, 158, 178, 177, 50, 117, 25, 61,255, 53, 138, 126, 109, 84, 198, 128, 195, 189, 13, 87, 223, 245, 36, 169, 62, 168, 67, 201,215, 121,214, 246, 124, 34, 185, 3, 224, 15, 236, 222, 122, 148, 176, 188, 220, 232, 40, 80, 78, 51, 10, 74, 167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130, 100, 159, 38, 65, 173, 69, 70, 146, 39, 94, 85, 47, 140, 163, 165, 125, 105, 213, 149, 59, 7, 88, 179, 64, 134, 172, 29, 247, 48, 55, 107, 228, 136, 217, 231, 137, 225, 27, 131,73, 76, 63, 248, 254, 141,83, 170, 144, 202, 216, 133, 97, 32, 113, 103, 164, 45, 43, 9, 91,203, 155, 37, 208, 190, 229, 108, 82, 89, 166, 116, 210, 230, 244, 180, 192, 209, 102, 175, 194, 57, 75, 99, 182)

$\pi : \mathbb{Z}_{256} \to \mathbb{Z}_{256}$, подстановка

$$S^{-1} : V_{128} \to V_{128} : S^{-1}(a) = \pi^{-1}(a_{15})||...\pi^{-1}(a_0)$$

Тут π^{-1} является обратной подстановкой к π, а S^{-1} обратным отображением к S

$$l : V_8^{16} \to V_8 : l(a_{15}, ..., a_0) = \nabla(148 \cdot \Delta(a_{15}) + 32 \cdot \Delta(a_{14}) +$$

$$+133 \cdot \Delta(a_{13}) + 16 \cdot \Delta(a_{12}) + 194 \cdot \Delta(a_{11}) + 192 \cdot \Delta(a_{10}) + 1 \cdot \Delta(a_9) + 251 \cdot \Delta(a_8) +$$
$$+1 \cdot \Delta(a_7) + 192 \cdot \Delta(a_6) + 194 \cdot \Delta(a_5) + 16 \cdot \Delta(a_4) + 133 \cdot \Delta(a_3) + 32 \cdot \Delta(a_2) +$$
$$148 \cdot \Delta(a_1) + 1 \cdot \Delta(a_0))$$

$\Delta : V_8 \to \mathbb{Z}_2/x^8 + x^7 + x^6 + x + 1$ переводит 8-ми битную строку в элемент конечного поля например:

$\Delta(15) = \Delta(00001111_2) = 0 \cdot x^7 + 0 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x + 1 \cdot 1 =$
$= x^3 + x^2 + x + 1$
$\mathbb{Z}_2[x]/x^8 + x^7 + x^6 + x + 1$ является конечным полем мощностью 256, т.к многочлен $x^8 + x^7 + x^6 + x + 1$ является неприводимым над $\mathbb{Z}_2[x]$
$\nabla : \mathbb{Z}_2[x]/x^8 + x^7 + x^6 + x + 1 \to V_8$ является обратным отображением к Δ

Таким образом, отображение l является получением 8 битной строки путем суммирования и умножения на константы элементов конечного поля, полученных от 16 байт 128 битной строки a

$R : V_{128} \to V_{128} : R(a) = R(a_{15}||...||a_0) = l(a_{15}, ..., a_0)||a_{15}||...||a_1$ Такое преобразование проще изобразить схемой:

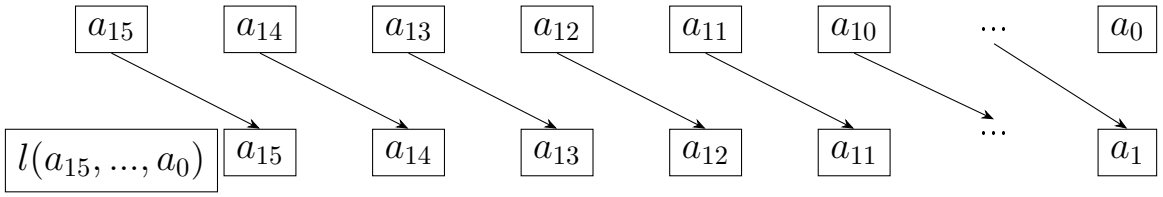

Схема 1. Преобразование R

Получается: a_{15} будет равен $l(a_{15}, ..., a_0)$, для других $a_i = a_{i+1}, i = 14, ...0$

$$L : V_{128} \to V_{128} : L(a) = R^{16}(a)$$

На строку a 16 раз применяется преобразование R. Таким образом на выходе строка a будет заполнена 16 байтами, которые будут отличны от исходной строки a

$R^{-1} : V_{128} \to V_{128} : R^{-1}(a) = R^{-1}(a_{15}, ..., a_0) =$
$a_{14}||a_{13}||...||a_0||l(a_{14}, a_{13}, ..., a_0, a_{15})$
Преобразование проще изобразить схемой:

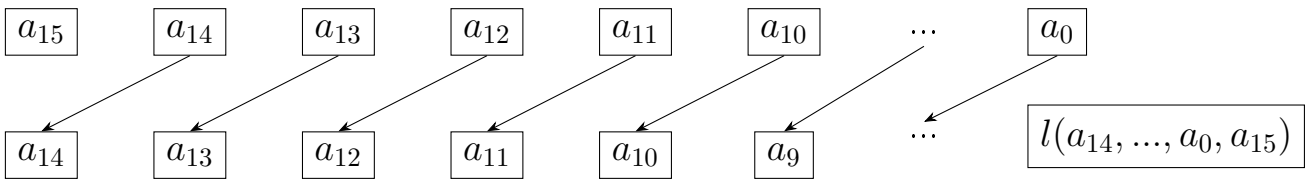

Схема 2. Преобразование R^{-1}

Получается: a_0 будет равен $l(a_{14}, ..., a_0, a_{15})$, для других $a_i = a_{i-1}, i = 15, ...1$

$$L^{-1} : V_{128} \to V_{128} : L^{-1}(a) = (R^{-1})^{16}(a)$$

Применяем на строку a 16 преобзований R^{-1}

$$F[k] : V_{128} \times V_{128} \to V_{128} \times V_{128} : F[k](a_1, a_0) = (LSX[k](a_1) \oplus a_0, a_1)$$

изображу его на схеме:

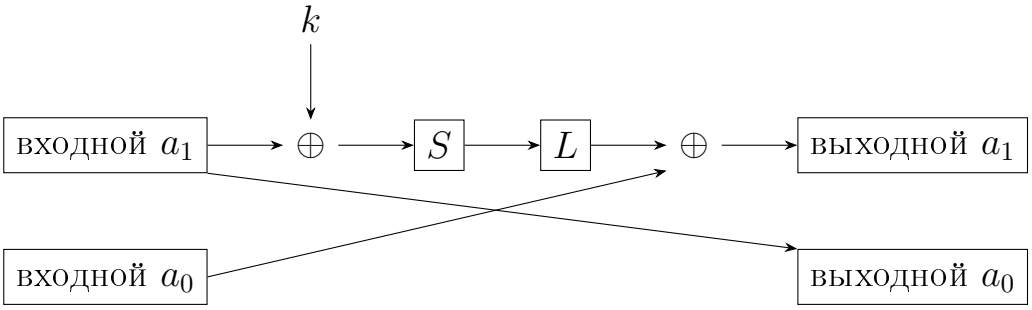

Схема 3. Преобразование $F[k]$

Алгоритм развертывания ключа

Алгоритм развертывания ключа использует итерационные константы $C_i \in V_{128}, i = 1, 2, ..., 32$, которые определены следующим образом:
$C_i = L(0\text{x}00||0\text{x}00||...||0\text{x}00||\text{i})$ т.к $\forall i < 256$ то $C_i[0] = i$
Преобразование L описывалось выше
Итерационные ключи $K_i \in V_{128}, i = 1, 2, ..10$ вырабатываются на основе ключа $K = K_{255}||...||K_0 \in V_{256}$ и определяются равенствами:

$$K_1 = K_{255}||...||K_{128}, K_2 = K_{127}||...||K_0$$
$$(K_{2i+1}, K_{2i+2}) = F[C_{8(i-1)+8}]...F[C_{8(i-1)+1}](K_{2i-1}, K_{2i}), i = 1, 2, 3, 4$$

Применяем 8 преобразований F с 8-мью константами C_i, чтобы получить пару ключей $K_{2i+1}K_{2i+2}$

Базовый алгоритм шифрования

после того как сработал алгоритм развертывания ключа, можно применить шифрование E:

$$E_{K_1,...,K_{10}} = X[K_{10}]LSX[K_9]...LSX[K_2]LSX[K_1](a)$$

изображу на схеме:

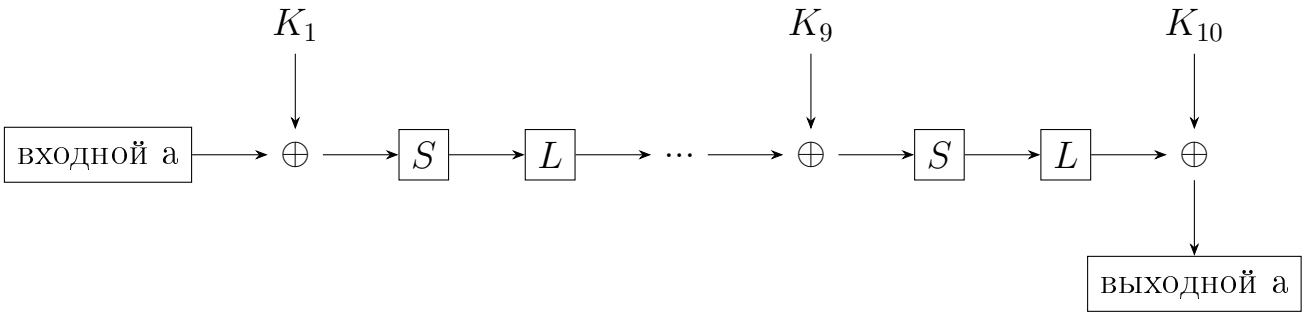

Схема 4. Алгоритм шифрования
Алгоритм расшифрования

После того как сработал алгоритм развертывания ключа, можно применить расшифрование D:

$$D_{K_1,\ldots,K_{10}}(a) = X[K_1]S^{-1}L^{-1}X[K_2]\ldots S^{-1}L^{-1}X[k_9]S^{-1}L^{-1}X[K_{10}](a)$$

изображу на схеме:

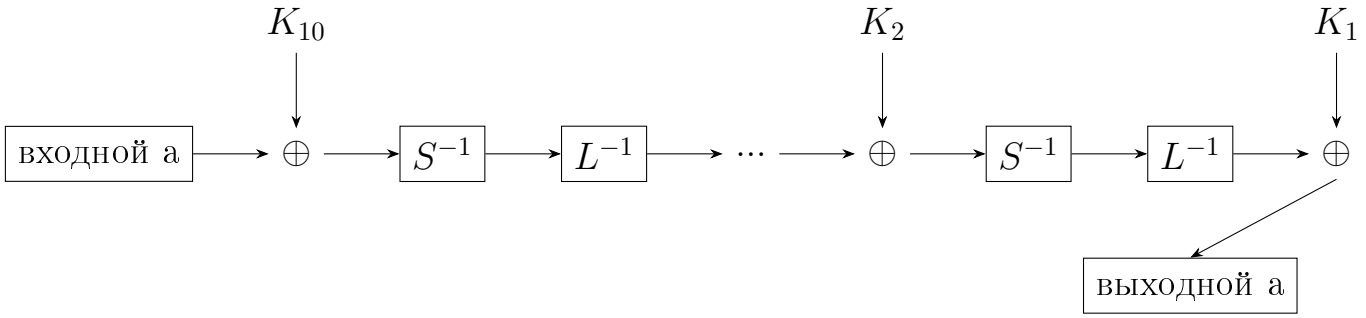

Схема 5. Алгоритм расшифрования

1.2. режим простой замены с зацеплением (CBC)

Режим шифрования — метод применения блочного шифра (алгоритма), позволяющий преобразовать последовательность блоков открытых данных в последовательность блоков зашифрованных данных. При этом для шифрования одного блока могут использоваться данные другого блока.

Шифрование в режиме простой замены с зацеплением.

Сообщение разбивается на блоки одинакового размера. Размер (длина) блока равен n и измеряется в битах. При необходимости последний блок дополняется до длины n. В нашем случае n=128
P_i— блок открытого текста
C_i— блок закрытого текста
E_k—шифрование определенным алгоритмом по ключу k
IV—синхропосылка, которая задается случайным образом(в идеале для шифрования разных сообщений лучше использовать разные синхропосылки)
$C_i = E_k(P_i \oplus C_{i-1})$
$C_0 = IV$
Изображу на схеме:

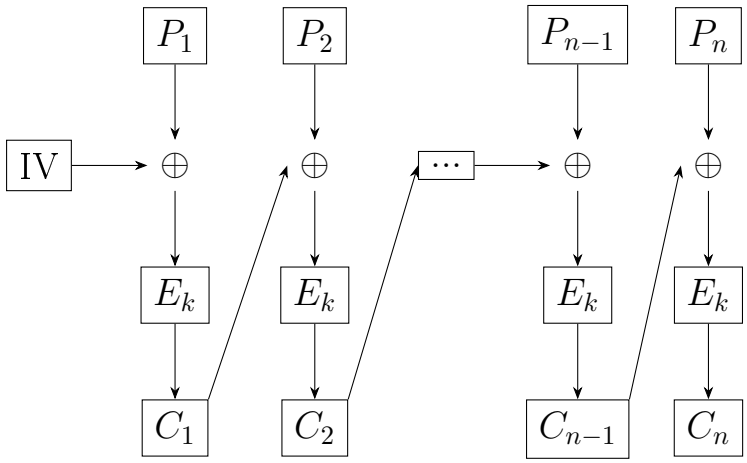

Схема 6. Шифрование CBC

Расшифрование CBC

$P_i = C_{i-1} \oplus D_k(C_i, k), C_0 = IV$

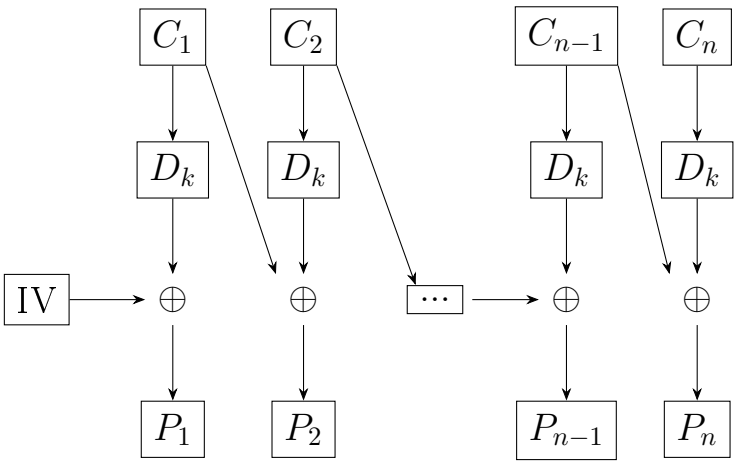

Схема 7. Расшифрование CBC

2. Практическая часть

2.1. реализация ГОСТ 34.12-2018 «Кузнечик»

Решил использовать язык программирования C и ОС Linux. Для начала обьявил константы:

```
  uint8_t PI[256] = { 252, 238, 221, 17, 207, 110, 49, 22, 251,
     196, 250, 218, 35, 197, 4, 77, 233, 119, 240, 219, 147,
46, 153, 186, 23, 54, 241, 187, 20, 205, 95, 193, 249, 24, 101,
   90, 226, 92, 239, 33, 129, 28, 60, 66,
139, 1, 142, 79, 5, 132, 2, 174, 227, 106, 143, 160, 6, 11,237,
   152, 127, 212, 211,31,235, 52, 44, 81,
234, 200, 72, 171,242, 42, 104, 162, 253, 58, 206, 204, 181, 112,
    14, 86, 8, 12, 118, 18, 191, 114, 19,
71, 156, 183, 93, 135, 21,161, 150, 41, 16, 123, 154, 199, 243,
   145, 120, 111, 157, 158, 178, 177, 50,
117, 25, 61,255, 53, 138, 126, 109, 84, 198, 128, 195, 189, 13,
   87, 223, 245, 36, 169, 62, 168, 67,
201,215, 121,214, 246, 124, 34, 185, 3, 224, 15, 236, 222, 122,
   148, 176, 188, 220, 232, 40, 80, 78,
51, 10, 74, 167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130,
   100, 159, 38, 65, 173, 69, 70, 146, 39,
94, 85, 47, 140, 163, 165, 125, 105, 213, 149, 59, 7, 88, 179,
   64, 134, 172, 29, 247, 48, 55, 107, 228,
136, 217, 231, 137, 225, 27, 131,73, 76, 63, 248, 254, 141,83,
   170, 144, 202, 216, 133, 97, 32, 113,
103, 164, 45, 43, 9, 91,203, 155, 37, 208, 190, 229, 108, 82, 89,
    166, 116, 210, 230, 244, 180, 192,
209, 102, 175, 194, 57, 75, 99, 182 };

uint8_t consts_for_l[16] =
{ 1,148,32,133,
16,194,192,1,
251,1,192,194,
16,133,32,148 };

uint8_t C[32][16];

uint8_t keys[10][16];
```

Листинг 1. Обьявление констант

Затем реализовал преобразования:

```
  void X(uint8_t* a, uint8_t* b)
  {
    for (int i = 0; i < 16; i++)
```

```
    {
      a[i] ^= b[i];
    }
  }
```

Листинг 2. Реализация преобразования X

В преобразовании X на вход дается 2 массива(заранее известно, что их размер равен 16 байтам)
По окончании вызова функции полученные значения будут храниться в массиве a

```
void S(uint8_t* a)
{
  for (int i = 0; i < 16; i++)
  {
    a[i] = PI[a[i]];
  }
}
```

Листинг 3. Реализация преобразования S

Т.к подстановка хранится в массиве, то отображение байтов будет проходить по значению по индексу массива

```
void r_S(uint8_t* a)
{
  for (int i = 0; i < 16; i++)
  {
    for (int j = 0; j < 256; j++)
    {
      if (a[i] == PI[j])
      {
        a[i] = j;
        break;
      }
    }
  }
}
```

Листинг 4. Реализация преобразования S^{-1}

обратным элементом подстановки π является индекс массива, поэтому я прохожусь по всем значениям массива, пока не найду одинаковый элемент и присваиваю a[i]=j
Для реализации преобразования R понадобится создать функцию, которая умножает элементы конечного поля

```
uint8_t GF_mul(uint8_t a, uint8_t b)
{
  uint8_t c = 0;
  uint8_t flag;
  for (int i = 0; i < 8; i++)
  {
    if (b == 1)
    {
      c ^= a;
      break;
    }
    flag = a & 0x80;
    a <<= 1;
    if ((int8_t)flag < 0)
    {
      a ^= 0xc3;
    }
    b >>= 1;
  }
  return c;
}
```

Листинг 5. умножение в $\mathbb{Z}_2/x^8 + x^7 + x^6 + x + 1$

при умнножении многочлена на многочлен происходит повышение степени, что эквивалентно битовому сдвигу слева. Т.к максимальная степень элемента конечного поля не должна превышать седьмой степени, то нужно после каждого битового сдвига проверять на наличие старшего бита в числе, если такой бит присутствует, то просто ксорим его с порождающим многочленом(его представлениее в виде числа 0xc3). Конечный будет результат тогда, когда второй элемент(на который мы умножаем первый элемент) после битовго сдвига вправо будет равен единице(т.к при умножении на свободный коэфицент в $\mathbb{Z}_2[x]$) степень многчлена не повысится.

```
uint8_t l(uint8_t* a)
{
  uint8_t S = 0;
  for (int i = 0; i < 16; i++)
  {
    S ^= GF_mul(a[i],consts_for_l[i]);
  }
  return S;
}
```

Листинг 6. реализация l преобразования

```
void R(uint8_t* a)
{
  uint8_t s = l(a);
  for (int i = 0; i < 15;i++)
  {
    a[i] = a[i + 1];
  }
  a[15] = s;
}
```

Листинг 7. реализация R преобразования

Действовал согласно схеме 1.

```
void L(uint8_t* a)
{
  for (int i = 0; i < 16; i++)
  {
    R(a);
  }
}
```

Листинг 8. реализация L преобразования

```
void r_R(uint8_t* a)
{
  uint8_t temp[16];
  for (int i = 15; i > 0; i--)
  {
    temp[i] = a[i - 1];
  }
  temp[0] = a[15];
  for (int i = 15; i > 0; i--)
  {
    a[i] = a[i - 1];
  }
  a[0] = l(temp);
}
```

Листинг 9. реализация R^{-1} преобразования

Действовал согласно схеме 2. Единственное, что пришлось сделать-это создать буферный массив, чтобы прогнать преобразования l т.к написано в госте для преобразования R^{-1}

```
void r_L(uint8_t* a)
{
  for (int i = 0; i < 16; i++)
  {
    r_R(a);
  }
}
```

Листинг 10. реализация L^{-1} преобразования

```
void F(uint8_t* C, uint8_t* a1, uint8_t* a0)
{
  uint8_t temp[16];
  for (int i = 0; i < 16; i++)
  {
    temp[i] = a1[i];
  }
  X(a1,C);
  S(a1);
  L(a1);
  X(a1, a0);
  for (int i = 0; i < 16; i++)
  {
    a0[i] = temp[i];
  }
}
```

Листинг 11. реализация F преобразования

Действовал согласно схеме 3. Создал буферный массив, чтобы сохранить входные значения a1 и передать их на выходной a0

```
void get_C()
{
  for (int i = 1; i < 33; i++)
  {
    C[i - 1][0] = i;
    L(C[i - 1]);
  }
}
```

Листинг 12. получение C_i

```
void get_keys(uint8_t* KEY)
{
  get_C();
  for (int i = 0; i < 16; i++)
  {
    keys[1][i] = KEY[i];
  }
  for (int i = 16; i < 32; i++)
  {
    keys[0][i - 16] = KEY[i];
  }
  uint8_t temp[2][16];
  for (int i = 0; i < 4; i++)
  {
    for (int j = 0; j < 16; j++)
    {
      temp[0][j] = keys[2*i][j];
      temp[1][j] = keys[2 * i + 1][j];
    }
    for (int k = 0; k < 8; k++)
    {
      F(C[8 * i + k], temp[0], temp[1]);
    }
    for (int m = 0; m < 16; m++)
    {
      keys[2 * i + 2][m] = temp[0][m];
      keys[2 * i + 3][m] = temp[1][m];
    }
  }
}
```

Листинг 13. получение ключей

Трудность возникла только в индексации т.к в госте отсчет идет с 1

```
void E(uint8_t* a)
{
  //get_keys(key);
  for (int i = 0; i < 9; i++)
  {
    X(a, keys[i]);
    S(a);
    L(a);
  }
  X(a, keys[9]);
}
```

Листинг 14. шифрование

Действовал согласно схеме 4.

```
void D(uint8_t* a)
{
  //get_keys(key);
  for (int i = 9; i > 0; i--)
  {
    X(a, keys[i]);
    r_L(a);
    r_S(a);
  }
  X(a, keys[0]);
}
```

Листинг 15. расшифрование

Действовал согласно схеме 5.

Проверка алгоритма по контрольным значениям

Сверюсь с данными контрольных примеров госта только для алгоритма развертывания ключа и констант,шифрования и расшифрования, т.к в этих алгоритмах задействованы все преобразования описанные выше.
Результаты для сверки:
$C_1 = 6ea276726c487ab85d27bd10dd849401$
$C_2 = dc87ece4d890f4b3ba4eb92079cbeb02$
$C_3 = b2259a96b4d88e0be7690430a44f7f03$
$C_4 = 7bcd10733257914012551504$
$C_5 = 156f6d791fab511deabb0c502fd18105$
$C_6 = a74af7efab73df160dd208608b9efe06$
$C_7 = C9e8819dc73ba5ae50f5b570561a6a07$
$C_8 = f6593616e6055689adfba18027aa2a08$
$K = 8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef$
$K_1 = 8899aabbccddeeff0011223344556677$
$K_2 = fedcba98765432100123456789abcdef$
$K_3 = db31485315694343228d6aef8cc78c44$
$K_4 = 3d4553d8e9cfec6815ebadc40a9ffd04$
$K_5 = 57646468c44a5e28d3e59246f429f1ac$
$K_6 = bd079435165c6432b532e82834da581b$
$K_7 = 51e640757e8745de705727265a0098b1$
$K_8 = 5a7925017b9fdd3ed72a91a22286f984$

$K_9 = bb44e25378c73123a5f32f73cdb6e517$
$K_{10} = 72e9dd7416bcf45b755dbaa88e4a4043$
Код для сверки будет в приложении 3.

2.2. Реализация режима простой замены с зацеплением

```
int get_IV(uint8_t* IV)
{
  int FD=open("/dev/urandom",O_RDONLY);
  if(FD<0)
  {
    printf("FILE /dev/urandom didnt open\n");
    return -1;
  }
  if(read(FD,IV,16)<0)
  {
    printf("Can't read file\n");
    close(FD);
    return -2;
  }
  close(FD);
  return 0;
}
```

Листинг 16. получение синхропосылки

Синхропосылку получал из файла /dev/urandom, т.к в него поступают шумы ОС, которые сложно предугадать.

```
void CBC(uint8_t* a,uint8_t* IV,uint8_t m)
{
  if(m)
  {
    uint8_t temp[16];
    copy(temp,a,16);
    D(a);
    X(a,IV);
    copy(IV,temp,16);
  }
  else
  {
    X(a,IV);
    E(a);
    copy(IV,a,16);
  }
}
```

Листинг 17. режим простой замены с зацеплением

Следовал схемам 6 и 7. функция copy(uint8_t* a,uint8_t b,int n) копирует значения массива b в массив a размером n

2.3. Реализация приложения

На вход программе подается ключ и название файла. В своей программе я сделал проверку на колво введенных парметров, 1-ый параметр проходит проверку на ключ(равна ли его длина 32байтам, причем если не равна,то открываю файл с названием первого парамета и считываю оттуда 32 байта, если уже и там нет данных для ключа(или такого файла не существует),то закрываю программу)
2 параметр это название файла, который надо будет шифровать или расшифровать, моя программа проверяет на наличие такого файла, если в названии есть начальные 2 символа "E_ то приступает к расшифрованию, если нет, то шифрует файл и именует его в "E_<название файла>"
полный код можно посмотреть в приложении 2.
хочу обратить внимание на дополнение последнего блока файла при шифровании и занесении синхропосылки в шифрованный файл.

```
    while((k=read(FD,blk,16))==16)
      {
        CBC(blk,IV,0);
        write(F,blk,16);
      }
      for(int i=k;i<16;i++)
      {
        blk[i]=16-k;
      }
      CBC(blk,IV,0);
      write(F,blk,16);
      close(F);
      close(FD);
      printf("ENCRYPTION COMPLETED\n");
```

Листинг 18. дополнение последнего блока

Последний блок дополняется 16-k байтами, значение каждого дополненого байта будет равно 16-k, при k=0 все равно дополняю блоком в котором все байты будут равны 16
Синхропосылку я генерирую, шифрую по E_k и записываю в начало файла, при расшифровании считываю первые 16 байт, расшифровываю и таким образом узнаю синхропосылку

Заключение

В ходе выполнения курсовой работы, мне пригодились знания, которые я получил за предыдущие курсы, особенно знания в алгебре и программировании. Кроме того, данная работа вызвала заинтересованность в моей будущей специальности.

Список использованной литературы

1. ГОСТ 34.12-2018

2. А.П. Алферов, А.Ю. Зубов, А.С. Кузьмин, А.В. Черемушкин "Основы криптографии"

3. Шнайер Б. – Прикладная криптография. Протоколы, алгоритмы и исходные тексты на языке С

3. Приложения

3.1. Приложение 1.ГОСТ 34.12-2018 «Кузнечик»

```
#include <stdio.h>
#include <stdint.h>

uint8_t PI[256] = { 252, 238, 221, 17, 207, 110, 49, 22, 251,
   196, 250, 218, 35, 197, 4, 77, 233, 119, 240, 219, 147,
46, 153, 186, 23, 54, 241, 187, 20, 205, 95, 193, 249, 24, 101,
   90, 226, 92, 239, 33, 129, 28, 60, 66,
139, 1, 142, 79, 5, 132, 2, 174, 227, 106, 143, 160, 6, 11,237,
   152, 127, 212, 211,31,235, 52, 44, 81,
234, 200, 72, 171,242, 42, 104, 162, 253, 58, 206, 204, 181, 112,
    14, 86, 8, 12, 118, 18, 191, 114, 19,
71, 156, 183, 93, 135, 21,161, 150, 41, 16, 123, 154, 199, 243,
   145, 120, 111, 157, 158, 178, 177, 50,
117, 25, 61,255, 53, 138, 126, 109, 84, 198, 128, 195, 189, 13,
   87, 223, 245, 36, 169, 62, 168, 67,
201,215, 121,214, 246, 124, 34, 185, 3, 224, 15, 236, 222, 122,
   148, 176, 188, 220, 232, 40, 80, 78,
51, 10, 74, 167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130,
   100, 159, 38, 65, 173, 69, 70, 146, 39,
94, 85, 47, 140, 163, 165, 125, 105, 213, 149, 59, 7, 88, 179,
   64, 134, 172, 29, 247, 48, 55, 107, 228,
136, 217, 231, 137, 225, 27, 131,73, 76, 63, 248, 254, 141,83,
   170, 144, 202, 216, 133, 97, 32, 113,
103, 164, 45, 43, 9, 91,203, 155, 37, 208, 190, 229, 108, 82, 89,
    166, 116, 210, 230, 244, 180, 192,
209, 102, 175, 194, 57, 75, 99, 182 };

uint8_t consts_for_l[16] =
{ 1,148,32,133,
16,194,192,1,
251,1,192,194,
16,133,32,148 };

uint8_t C[32][16];

uint8_t keys[10][16];

void X(uint8_t* a, uint8_t* b)
{
  for (int i = 0; i < 16; i++)
  {
    a[i] ^= b[i];
  }
```

```
}

void S(uint8_t* a)
{
  for (int i = 0; i < 16; i++)
  {
    a[i] = PI[a[i]];
  }
}

void r_S(uint8_t* a)
{
  for (int i = 0; i < 16; i++)
  {
    for (int j = 0; j < 256; j++)
    {
      if (a[i] == PI[j])
      {
        a[i] = j;
        break;
      }
    }
  }
}

uint8_t GF_mul(uint8_t a, uint8_t b)
{
  uint8_t c = 0;
  uint8_t flag;
  for (int i = 0; i < 8; i++)
  {
    if ((b & 1) == 1)
    {
      c ^= a;
    }
    flag = a & 0x80;
    a <<= 1;
    if ((int8_t)flag < 0)
    {
      a ^= 0xc3;
    }
    b >>= 1;
  }
  return c;
}

uint8_t l(uint8_t* a)
```

```
{
  uint8_t S = 0;
  for (int i = 0; i < 16; i++)
  {
    S ^= GF_mul(a[i],consts_for_l[i]);
  }
  return S;
}

void R(uint8_t* a)
{
  uint8_t s = l(a);
  for (int i = 0; i < 15;i++)
  {
    a[i] = a[i + 1];
  }
  a[15] = s;
}

void L(uint8_t* a)
{
  for (int i = 0; i < 16; i++)
  {
    R(a);
  }
}

void r_R(uint8_t* a)
{
  uint8_t temp[16];
  for (int i = 15; i > 0; i--)
  {
    temp[i] = a[i - 1];
  }
  temp[0] = a[15];
  for (int i = 15; i > 0; i--)
  {
    a[i] = a[i - 1];
  }
  a[0] = l(temp);
}

void r_L(uint8_t* a)
{
  for (int i = 0; i < 16; i++)
  {
    r_R(a);
```

```
127   }
128 }
129
130 void get_C()
131 {
132   uint8_t temp[16];
133   uint8_t t;
134   for (int i = 1; i < 33; i++)
135   {
136     C[i - 1][0] = i;
137     L(C[i - 1]);
138   }
139 }
140
141 void F(uint8_t* C, uint8_t* a1, uint8_t* a0)
142 {
143   uint8_t temp[16];
144   for (int i = 0; i < 16; i++)
145   {
146     temp[i] = a1[i];
147   }
148   X(a1,C);
149   S(a1);
150   L(a1);
151   X(a1, a0);
152   for (int i = 0; i < 16; i++)
153   {
154     a0[i] = temp[i];
155   }
156
157 }
158
159 void get_keys(uint8_t* KEY)
160 {
161   get_C();
162   for (int i = 0; i < 16; i++)
163   {
164     keys[1][i] = KEY[i];
165   }
166   for (int i = 16; i < 32; i++)
167   {
168     keys[0][i - 16] = KEY[i];
169   }
170   uint8_t temp[2][16];
171   for (int i = 0; i < 4; i++)
172   {
173     for (int j = 0; j < 16; j++)
```

```
174       {
175         temp[0][j] = keys[2*i][j];
176         temp[1][j] = keys[2 * i + 1][j];
177       }
178       for (int k = 0; k < 8; k++)
179       {
180         F(C[8 * i + k], temp[0], temp[1]);
181       }
182       for (int m = 0; m < 16; m++)
183       {
184         keys[2 * i + 2][m] = temp[0][m];
185         keys[2 * i + 3][m] = temp[1][m];
186       }
187     }
188 
189 }
190 
191 void E(uint8_t* a)
192 {
193   //get_keys(key);
194   for (int i = 0; i < 9; i++)
195   {
196     X(a, keys[i]);
197     S(a);
198     L(a);
199   }
200   X(a, keys[9]);
201 }
202 
203 void D(uint8_t* a)
204 {
205   //get_keys(key);
206   for (int i = 9; i > 0; i--)
207   {
208     X(a, keys[i]);
209     r_L(a);
210     r_S(a);
211   }
212   X(a, keys[0]);
213 }
```

3.2. Приложение 2.Режим простой замены с зацеплением и само консольное приложение

```
#include "GOST.h"
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
void copy(uint8_t* a,uint8_t* b,int n)
{
  for(int i=0;i<n;i++)
  {
    a[i]=b[i];
  }
}

int get_IV(uint8_t* IV)
{
  int FD=open("/dev/urandom",O_RDONLY);
  if(FD<0)
  {
    printf("FILE /dev/urandom didnt open\n");
    return -1;
  }
  if(read(FD,IV,16)<0)
  {
    printf("Can't read file\n");
    close(FD);
    return -2;
  }
  close(FD);
  return 0;
}

int CBC(uint8_t* a,uint8_t* IV,uint8_t m)
{
  if(m)
  {
    uint8_t temp[16];
    copy(temp,a,16);
    D(a);
    X(a,IV);
    copy(IV,temp,16);
  }
  else
  {
    X(a,IV);
```

```
45      E(a);
46      copy(IV,a,16);
47    }
48
49  }
50
51  int main(int argc,char* argv[])
52  {
53    if(argc!=3)
54    {
55      printf("Enter KEY and file");
56
57    }
58    int FD=open(argv[2],O_RDWR);
59    if(FD<0)
60    {
61      printf("cant open your file\n");
62      return 0;
63    }
64    uint8_t key[32];
65    if(strlen(argv[1])!=32)
66    {
67      int FD_key=open(argv[1],O_RDONLY);
68      if(FD_key<0)
69      {
70        printf("Your key is short and its not a file\n");
71        return 0;
72      }
73      if(read(FD_key,key,32)!=32)
74      {
75        printf("key in file is short\n");
76        return 0;
77      }
78      close(FD_key);
79    }
80    else
81    {
82      copy(key,argv[1],32);
83    }
84    get_keys(key);
85    uint8_t IV[16];
86    uint8_t blk[16];
87    if(argv[2][0]=='E' && argv[2][1]=='_')
88    {
89      int F=creat(argv[2]+2,0666);
90      if(F<0)
91      {
```

```
92        printf("file didnt creat\n");
93        return 0;
94      }
95      read(FD,IV,16);
96      D(IV);
97      int k;
98      while((k=read(FD,blk,16))==16)
99      {
100       if((k=read(FD,blk,16))!=0)
101       {
102         lseek(FD,-32,SEEK_CUR);
103         k=read(FD,blk,16);
104         CBC(blk,IV,1);
105         write(F,blk,16);
106       }
107       else
108       {
109         lseek(FD,SEEK_CUR,-16);
110         k=read(FD,blk,16);
111         CBC(blk,IV,1);
112         if(blk[15]!=16)
113           write(F,blk,16-blk[15]);
114       }
115
116     }
117     close(F);
118     close(FD);
119     printf("DECRYPTION COMPLETED\n");
120   }
121   else
122   {
123     char* name=(char*)malloc(sizeof(char)*(2+strlen(argv[2])));
124     name[0]='E';
125     name[1]='_';
126     copy(name+2,argv[2],strlen(argv[2]));
127     int F=creat(name,0666);
128     free(name);
129     if(F<0)
130     {
131       printf("file didnt creat\n");
132       return 0;
133     }
134     if(get_IV(IV)<0)
135     {
136       printf("error in get_iv()\n");
137     }
138     uint8_t temp[16];
```

```
139       copy(temp,IV,16);
140       E(temp);
141       write(F,temp,16);
142       int k;
143       while((k=read(FD,blk,16))==16)
144       {
145         CBC(blk,IV,0);
146         write(F,blk,16);
147       }
148       for(int i=k;i<16;i++)
149       {
150
151         blk[i]=16-k;
152       }
153       CBC(blk,IV,0);
154       write(F,blk,16);
155       close(F);
156       close(FD);
157       printf("ENCRYPTION COMPLETED\n");
158    }
159    return 0;
160 }
```

3.3. Приложение 3.Проверка контрольных значений

```
#include "GOST.h"


void print_arr(uint8_t* a,int n)
{
    for(int i=n-1;i>-1;i--)
    {
        if(a[i]<0x10)
        {
            printf("0%x",a[i]);
        }
        else
        {
            printf("%x",a[i]);
        }

    }
    printf("\n");
}


int main(int argc,char* argv[])
{
    uint8_t key[32]=
    {0xef,0xcd,0xab,0x89,0x67,0x45,0x23,0x01,
    0x10,0x32,0x54,0x76,0x98,0xba,0xdc,0xfe,
    0x77,0x66,0x55,0x44,0x33,0x22,0x11,0x00,
    0xff,0xee,0xdd,0xcc,0xbb,0xaa,0x99,0x88};
    get_keys(key);
    uint8_t a[16]=
    {0x88,0x99,0xaa,0xbb,0xcc,0xdd,0xee,0xff,
    0x00,0x77,0x66,0x55,0x44,0x33,0x22,0x11};
    for(int i=0;i<32;i++)
    {
        printf("C[%d]=",i+1);
        print_arr(C[i],16);
    }
    printf("KEY=");
    print_arr(key,32);
    for(int i=0;i<10;i++)
    {
        printf("K[%d]=",i+1);
        print_arr(keys[i],16);
    }
    printf("a=");
```

```
46      print_arr(a,16);
47      E(a);
48      printf("b=");
49      print_arr(a,16);
50      D(a);
51      printf("a=");
52      print_arr(a,16);
53      return 0;
54  }
```