

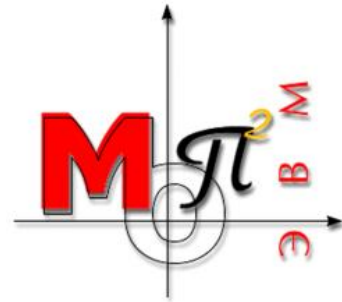
МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего  
образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ



## ЛАБОРАТОРНАЯ РАБОТА № 6

по дисциплине

**«Безопасность информационных технологий»**

на тему:

**«Алгоритм шифрования Кузнечик»**

Выполнил:

Студент группы

КТбо2-8

Нестеренко П. А.

Проверил:

доцент кафедры

ИБТКС

Петров Д. А.

Оценка

---

« \_\_\_\_ » \_\_\_\_\_ 2020 г.

## **Цель работы**

Исследование принципа работы симметричного блочного криптоалгоритма «Кузнечик» путём редактирования исходного кода криптоалгоритма и его пошаговой отладки.

## **Порядок выполнения**

1. Ознакомиться с описанием блочного криптоалгоритма «Кузнечик» (Российского стандарта блочного шифрования, [www.tk26.ru](http://www.tk26.ru))
2. Ознакомиться с исходным кодом блочного криптоалгоритма «Кузнечик», в качестве примера дана реализация криптоалгоритма на языке Си от известного криптографа Markku-Juhani O. Saarinen (файлы `main.c`, `kuznechik.h`, `kuznechik_8bit.c`).
3. Скорректировать файл «`kuznechik_8bit.c`» для добавления команд вывода на экран отладочной информации с результатами выполнения каждой операции криптоалгоритма (операции подстановки, линейных функций L и R, раунда шифрования).
4. Скомпилировать программу командой «`gcc -Wall -Ofast -march=native main.c kuznechik_8bit.c -o xtest`»
5. Протестировать полученную программу и проанализировать результаты выполнения каждой операции криптоалгоритма (операции подстановки, линейных функций L и R, раунда шифрования).
6. В качестве отчёта по лабораторной работе привести результаты выполнения указанных выше пунктов, в том числе скорректированный исходный код и результаты работы программы.

## Ход работы

В исходный код были внесены правки для большего понимания принципа работы алгоритма, был убран код самотестирования алгоритма и проверки времени его работы. Результат выполнения программы приведен ниже. Скорректированный код находится в соответствующем разделе в приложении А.

Результат работы программы:

Установка 32 байтного ключа для шифрования...

Список раундовых ключей:

```
K_1   = 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77
K_2   = FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF
K_3   = DB 31 48 53 15 69 43 43 22 8D 6A EF 8C C7 8C 44
K_4   = 3D 45 53 D8 E9 CF EC 68 15 EB AD C4 0A 9F FD 04
K_5   = 57 64 64 68 C4 4A 5E 28 D3 E5 92 46 F4 29 F1 AC
K_6   = BD 07 94 35 16 5C 64 32 B5 32 E8 28 34 DA 58 1B
K_7   = 51 E6 40 75 7E 87 45 DE 70 57 27 26 5A 00 98 B1
K_8   = 5A 79 25 01 7B 9F DD 3E D7 2A 91 A2 22 86 F9 84
K_9   = BB 44 E2 53 78 C7 31 23 A5 F3 2F 73 CD B6 E5 17
K_10  = 72 E9 DD 74 16 BC F4 5B 75 5D BA A8 8E 4A 40 43
```

Исходные данные:

PT = 10 20 30 40 50 60 70 00 01 02 03 04 05 06 07 08

Шифрование ...

Шифрование

```
Исходный текст    10 20 30 40 50 60 70 00 01 02 03 04 05 06 07 08
Раундовый ключ (0) 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77
  Результат X   98 B9 9A FB 9C BD 9E FF 01 13 21 37 41 53 61 7F
  Результат PI  DC A3 28 C2 4E D5 0A B6 EE DB 18 A0 34 56 A1 57
  Результат L    92 50 F7 2E C9 B5 AC F4 00 50 CE 4C 19 38 8E 8C
Раундовый ключ (1) FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF
  Результат X   6C 8C 4D B6 BF E1 9E E4 01 73 8B 2B 90 93 43 63
  Результат PI  9D 7C 3A 55 3B 71 0A 2D EE 3D F6 42 E0 DE 51 29
  Результат L   B0 69 5F CF AD 5D 4E BD 1D 50 16 97 00 D1 63 30
Раундовый ключ (2) DB 31 48 53 15 69 43 43 22 8D 6A EF 8C C7 8C 44
  Результат X   6B 58 17 9C B8 34 0D FE 3F DD 7C 78 8C 16 EF 74
  Результат PI  6F BF BA 4E 8C E3 C5 63 1F D8 C3 6D 7C 99 52 FF
  Результат L   52 3B 7C 3D 5D FD 01 D8 4F 1A 11 69 8E F2 DC 01
Раундовый ключ (3) 3D 45 53 D8 E9 CF EC 68 15 EB AD C4 0A 9F FD 04
  Результат X   6F 7E 2F E5 B4 32 ED B0 5A F1 BC AD 84 6D 21 05
  Результат PI  B1 0D 4F 2B 27 02 E5 AD 13 A6 69 9F 3E 9E 18 6E
  Результат L   AE 5D 8B EB 09 D9 8A 1A 8C C0 D0 28 C1 9D 56 59
Раундовый ключ (4) 57 64 64 68 C4 4A 5E 28 D3 E5 92 46 F4 29 F1 AC
  Результат X   F9 39 EF 83 CD 93 D4 32 5F 25 42 6E 35 B4 A7 F5
  Результат PI  66 0B 52 A9 D9 DE 4C 02 87 5C 2C B2 6A 27 44 F4
  Результат L   3D 0C AD A6 51 B6 E7 2A 14 00 58 86 98 76 D8 67
Раундовый ключ (5) BD 07 94 35 16 5C 64 32 B5 32 E8 28 34 DA 58 1B
  Результат X   80 0B 39 93 47 EA 83 18 A1 32 B0 AE AC AC 80 7C
  Результат PI  DF DA 0B DE AB 25 A9 17 97 02 AD 26 64 64 DF C3
  Результат L   79 D0 D6 AC F3 F2 03 82 DD 0E 0B FE 1B C7 AE 95
Раундовый ключ (6) 51 E6 40 75 7E 87 45 DE 70 57 27 26 5A 00 98 B1
  Результат X   28 36 96 D9 8D 75 46 5C AD 59 2C D8 41 C7 36 24
  Результат PI  81 8F B0 53 22 35 48 9C 9F 72 8B 8D 34 F7 8F E2
  Результат L   57 48 A5 97 5D 99 B6 FD F8 E5 4A 45 F5 4D A0 97
Раундовый ключ (7) 5A 79 25 01 7B 9F DD 3E D7 2A 91 A2 22 86 F9 84
  Результат X   0D 31 80 96 26 06 6B C3 2F CF DB E7 D7 CB 59 13
  Результат PI  C5 84 DF B0 EF 31 6F 40 4F 89 90 5B FE E4 72 DB
  Результат L   BB 8C 7B 4A CE 34 A9 71 6B 02 2C 10 2F F9 05 BB
Раундовый ключ (8) BB 44 E2 53 78 C7 31 23 A5 F3 2F 73 CD B6 E5 17
```

Результат X 00 C8 99 19 B6 F3 98 52 CE F1 03 63 E2 4F E0 AC  
Результат PI FC 30 E8 36 55 D2 DC 0E E7 A6 11 29 67 CC 20 64  
Результат L 03 2B CA B8 27 9E 82 E7 EC 4D 89 E7 6C B9 32 AE  
Шифротекст: 03 2B CA B8 27 9E 82 E7 EC 4D 89 E7 6C B9 32 AE

Результат шифрования: 71 C2 17 CC 31 22 76 BC 99 10 33 4F E2 F3 72 ED  
Установка 32 байтного ключа для дешифрования...  
Дешифрование ...  
71 C2 17 CC 31 22 76 BC 99 10 33 4F E2 F3 72 ED

Результат обратного L FC 30 E8 36 55 D2 DC 0E E7 A6 11 29 67 CC 20 64  
Результат обратного PI 00 C8 99 19 B6 F3 98 52 CE F1 03 63 E2 4F E0 AC  
Раундовый ключ (8) BB 44 E2 53 78 C7 31 23 A5 F3 2F 73 CD B6 E5 17  
Результат X BB 8C 7B 4A CE 34 A9 71 6B 02 2C 10 2F F9 05 BB  
Результат обратного L C5 84 DF B0 EF 31 6F 40 4F 89 90 5B FE E4 72 DB  
Результат обратного PI 0D 31 80 96 26 06 6B C3 2F CF DB E7 D7 CB 59 13  
Раундовый ключ (7) 5A 79 25 01 7B 9F DD 3E D7 2A 91 A2 22 86 F9 84  
Результат X 57 48 A5 97 5D 99 B6 FD F8 E5 4A 45 F5 4D A0 97  
Результат обратного L 81 8F B0 53 22 35 48 9C 9F 72 8B 8D 34 F7 8F E2  
Результат обратного PI 28 36 96 D9 8D 75 46 5C AD 59 2C D8 41 C7 36 24  
Раундовый ключ (6) 51 E6 40 75 7E 87 45 DE 70 57 27 26 5A 00 98 B1  
Результат X 79 D0 D6 AC F3 F2 03 82 DD 0E 0B FE 1B C7 AE 95  
Результат обратного L DF DA 0B DE AB 25 A9 17 97 02 AD 26 64 64 DF C3  
Результат обратного PI 80 0B 39 93 47 EA 83 18 A1 32 B0 AE AC AC 80 7C  
Раундовый ключ (5) BD 07 94 35 16 5C 64 32 B5 32 E8 28 34 DA 58 1B  
Результат X 3D 0C AD A6 51 B6 E7 2A 14 00 58 86 98 76 D8 67  
Результат обратного L 66 0B 52 A9 D9 DE 4C 02 87 5C 2C B2 6A 27 44 F4  
Результат обратного PI F9 39 EF 83 CD 93 D4 32 5F 25 42 6E 35 B4 A7 F5  
Раундовый ключ (4) 57 64 64 68 C4 4A 5E 28 D3 E5 92 46 F4 29 F1 AC  
Результат X AE 5D 8B EB 09 D9 8A 1A 8C C0 D0 28 C1 9D 56 59  
Результат обратного L B1 0D 4F 2B 27 02 E5 AD 13 A6 69 9F 3E 9E 18 6E  
Результат обратного PI 6F 7E 2F E5 B4 32 ED B0 5A F1 BC AD 84 6D 21 05  
Раундовый ключ (3) 3D 45 53 D8 E9 CF EC 68 15 EB AD C4 0A 9F FD 04  
Результат X 52 3B 7C 3D 5D FD 01 D8 4F 1A 11 69 8E F2 DC 01  
Результат обратного L 6F BF BA 4E 8C E3 C5 63 1F D8 C3 6D 7C 99 52 FF  
Результат обратного PI 6B 58 17 9C B8 34 0D FE 3F DD 7C 78 8C 16 EF 74  
Раундовый ключ (2) DB 31 48 53 15 69 43 43 22 8D 6A EF 8C C7 8C 44  
Результат X B0 69 5F CF AD 5D 4E BD 1D 50 16 97 00 D1 63 30  
Результат обратного L 9D 7C 3A 55 3B 71 0A 2D EE 3D F6 42 E0 DE 51 29  
Результат обратного PI 6C 8C 4D B6 BF E1 9E E4 01 73 8B 2B 90 93 43 63  
Раундовый ключ (1) FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF  
Результат X 92 50 F7 2E C9 B5 AC F4 00 50 CE 4C 19 38 8E 8C  
Результат обратного L DC A3 28 C2 4E D5 0A B6 EE DB 18 A0 34 56 A1 57  
Результат обратного PI 98 B9 9A FB 9C BD 9E FF 01 13 21 37 41 53 61 7F  
Раундовый ключ (0) 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77  
Результат X 10 20 30 40 50 60 70 00 01 02 03 04 05 06 07 08

Расшифрованные данные:  
PT = 10 20 30 40 50 60 70 00 01 02 03 04 05 06 07 08

## Скорректированный исходный код программы:

### 1) main.c:

```
#include <stdio.h>
#include <time.h>
#include "kuznechik.h"
#include <locale.h>

void kuz_init();

int main(int argc, char **argv)
{
    setlocale(0, "Russian");

    const uint8_t testvec_key[32] = {
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10,
        0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF
    };
    const uint8_t testvec_pt[16] = {
        0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x00,
        0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08
    };

    int i;
    kuz_key_t key;
    w128_t x;

    printf("Установка 32 байтного ключа для шифрования...\n");
    kuz_set_encrypt_key(&key, testvec_key);

    printf("Список раундовых ключей:\n");
    for (i = 0; i < 10; i++) {
        printf("K_%d\t=", i + 1);
        print_w128(&key.k[i]);
    }

    printf("Исходные данные:\n");
    for (i = 0; i < 16; i++)
        x.b[i] = testvec_pt[i];
    printf("PT\t=");
    print_w128(&x);

    printf("Шифрование ... \n");
    kuz_encrypt_block(&key, &x);

    printf("Результат шифрования:\t");
    print_w128( (w128_t *) &x );

    printf("Установка 32 байтного ключа для дешифрования...\n");

    kuz_set_decrypt_key(&key, testvec_key);

    printf("Дешифрование ... \n");

    print_w128( (w128_t *) &x );
```

```

printf("\n");

kuz_decrypt_block(&key, &x);

printf("Расшифрованные данные:\n");
printf("PT =");
print_w128(&x);
}

```

## 2) kuznechik\_8bit.c

```

// kuznechik.c
// 04-Jan-15 Markku-Juhani O. Saarinen <mjos@iki.fi>

// This is the basic non-optimized 8-bit "readble" version of the cipher.

// Conforms to included doc/GOSTR-bsh.pdf file, which has an internal
// date of September 2, 2014.

#include <stdio.h>
#include "kuznechik.h"
#include <locale.h>

// The S-Box from section 5.1.1

const uint8_t kuz_pi[0x100] = {
    0xFC, 0xEE, 0xDD, 0x11, 0xCF, 0x6E, 0x31, 0x16, // 00..07
    0xFB, 0xC4, 0xFA, 0xDA, 0x23, 0xC5, 0x04, 0x4D, // 08..0F
    0xE9, 0x77, 0xF0, 0xDB, 0x93, 0x2E, 0x99, 0xBA, // 10..17
    0x17, 0x36, 0xF1, 0xBB, 0x14, 0xCD, 0x5F, 0xC1, // 18..1F
    0xF9, 0x18, 0x65, 0x5A, 0xE2, 0x5C, 0xEF, 0x21, // 20..27
    0x81, 0x1C, 0x3C, 0x42, 0x8B, 0x01, 0x8E, 0x4F, // 28..2F
    0x05, 0x84, 0x02, 0xAE, 0xE3, 0x6A, 0x8F, 0xA0, // 30..37
    0x06, 0x0B, 0xED, 0x98, 0x7F, 0xD4, 0xD3, 0x1F, // 38..3F
    0xEB, 0x34, 0x2C, 0x51, 0xEA, 0xC8, 0x48, 0xAB, // 40..47
    0xF2, 0x2A, 0x68, 0xA2, 0xFD, 0x3A, 0xCE, 0xCC, // 48..4F
    0xB5, 0x70, 0x0E, 0x56, 0x08, 0x0C, 0x76, 0x12, // 50..57
    0xBF, 0x72, 0x13, 0x47, 0x9C, 0xB7, 0x5D, 0x87, // 58..5F
    0x15, 0xA1, 0x96, 0x29, 0x10, 0x7B, 0x9A, 0xC7, // 60..67
    0xF3, 0x91, 0x78, 0x6F, 0x9D, 0x9E, 0xB2, 0xB1, // 68..6F
    0x32, 0x75, 0x19, 0x3D, 0xFF, 0x35, 0x8A, 0x7E, // 70..77
    0x6D, 0x54, 0xC6, 0x80, 0xC3, 0xBD, 0x0D, 0x57, // 78..7F
    0xDF, 0xF5, 0x24, 0xA9, 0x3E, 0xA8, 0x43, 0xC9, // 80..87
    0xD7, 0x79, 0xD6, 0xF6, 0x7C, 0x22, 0xB9, 0x03, // 88..8F
    0xE0, 0x0F, 0xEC, 0xDE, 0x7A, 0x94, 0xB0, 0xBC, // 90..97
    0xDC, 0xE8, 0x28, 0x50, 0x4E, 0x33, 0x0A, 0x4A, // 98..9F
    0xA7, 0x97, 0x60, 0x73, 0x1E, 0x00, 0x62, 0x44, // A0..A7
    0x1A, 0xB8, 0x38, 0x82, 0x64, 0x9F, 0x26, 0x41, // A8..AF
    0xAD, 0x45, 0x46, 0x92, 0x27, 0x5E, 0x55, 0x2F, // B0..B7
    0x8C, 0xA3, 0xA5, 0x7D, 0x69, 0xD5, 0x95, 0x3B, // B8..BF
    0x07, 0x58, 0xB3, 0x40, 0x86, 0xAC, 0x1D, 0xF7, // C0..C7
    0x30, 0x37, 0x6B, 0xE4, 0x88, 0xD9, 0xE7, 0x89, // C8..CF

```

```

    0xE1, 0x1B, 0x83, 0x49, 0x4C, 0x3F, 0xF8, 0xFE, // D0..D7
    0x8D, 0x53, 0xAA, 0x90, 0xCA, 0xD8, 0x85, 0x61, // D8..DF
    0x20, 0x71, 0x67, 0xA4, 0x2D, 0x2B, 0x09, 0x5B, // E0..E7
    0xCB, 0x9B, 0x25, 0xD0, 0xBE, 0xE5, 0x6C, 0x52, // E8..EF
    0x59, 0xA6, 0x74, 0xD2, 0xE6, 0xF4, 0xB4, 0xC0, // F0..F7
    0xD1, 0x66, 0xAF, 0xC2, 0x39, 0x4B, 0x63, 0xB6, // F8..FF
};

// Inverse S-Box

static const uint8_t kuz_pi_inv[0x100] = {
    0xA5, 0x2D, 0x32, 0x8F, 0x0E, 0x30, 0x38, 0xC0, // 00..07

    0x54, 0xE6, 0x9E, 0x39, 0x55, 0x7E, 0x52, 0x91, // 08..0F
    0x64, 0x03, 0x57, 0x5A, 0x1C, 0x60, 0x07, 0x18, // 10..17
    0x21, 0x72, 0xA8, 0xD1, 0x29, 0xC6, 0xA4, 0x3F, // 18..1F
    0xE0, 0x27, 0x8D, 0x0C, 0x82, 0xEA, 0xAE, 0xB4, // 20..27
    0x9A, 0x63, 0x49, 0xE5, 0x42, 0xE4, 0x15, 0xB7, // 28..2F
    0xC8, 0x06, 0x70, 0x9D, 0x41, 0x75, 0x19, 0xC9, // 30..37
    0xAA, 0xFC, 0x4D, 0xBF, 0x2A, 0x73, 0x84, 0xD5, // 38..3F
    0xC3, 0xAF, 0x2B, 0x86, 0xA7, 0xB1, 0xB2, 0x5B, // 40..47
    0x46, 0xD3, 0x9F, 0xFD, 0xD4, 0x0F, 0x9C, 0x2F, // 48..4F
    0x9B, 0x43, 0xEF, 0xD9, 0x79, 0xB6, 0x53, 0x7F, // 50..57
    0xC1, 0xF0, 0x23, 0xE7, 0x25, 0x5E, 0xB5, 0x1E, // 58..5F
    0xA2, 0xDF, 0xA6, 0xFE, 0xAC, 0x22, 0xF9, 0xE2, // 60..67
    0x4A, 0xBC, 0x35, 0xCA, 0xEE, 0x78, 0x05, 0x6B, // 68..6F
    0x51, 0xE1, 0x59, 0xA3, 0xF2, 0x71, 0x56, 0x11, // 70..77
    0x6A, 0x89, 0x94, 0x65, 0x8C, 0xBB, 0x77, 0x3C, // 78..7F
    0x7B, 0x28, 0xAB, 0xD2, 0x31, 0xDE, 0xC4, 0x5F, // 80..87
    0xCC, 0xCF, 0x76, 0x2C, 0xB8, 0xD8, 0x2E, 0x36, // 88..8F
    0xDB, 0x69, 0xB3, 0x14, 0x95, 0xBE, 0x62, 0xA1, // 90..97
    0x3B, 0x16, 0x66, 0xE9, 0x5C, 0x6C, 0x6D, 0xAD, // 98..9F
    0x37, 0x61, 0x4B, 0xB9, 0xE3, 0xBA, 0xF1, 0xA0, // A0..A7
    0x85, 0x83, 0xDA, 0x47, 0xC5, 0xB0, 0x33, 0xFA, // A8..AF
    0x96, 0x6F, 0x6E, 0xC2, 0xF6, 0x50, 0xFF, 0x5D, // B0..B7
    0xA9, 0x8E, 0x17, 0x1B, 0x97, 0x7D, 0xEC, 0x58, // B8..BF
    0xF7, 0x1F, 0xFB, 0x7C, 0x09, 0x0D, 0x7A, 0x67, // C0..C7
    0x45, 0x87, 0xDC, 0xE8, 0x4F, 0x1D, 0x4E, 0x04, // C8..CF
    0xEB, 0xF8, 0xF3, 0x3E, 0x3D, 0xBD, 0x8A, 0x88, // D0..D7
    0xDD, 0xCD, 0x0B, 0x13, 0x98, 0x02, 0x93, 0x80, // D8..DF
    0x90, 0xD0, 0x24, 0x34, 0xCB, 0xED, 0xF4, 0xCE, // E0..E7
    0x99, 0x10, 0x44, 0x40, 0x92, 0x3A, 0x01, 0x26, // E8..EF
    0x12, 0x1A, 0x48, 0x68, 0xF5, 0x81, 0x8B, 0xC7, // F0..F7
    0xD6, 0x20, 0x0A, 0x08, 0x00, 0x4C, 0xD7, 0x74 // F8..FF
};

// Linear vector from sect 5.1.2

static const uint8_t kuz_lvec[16] = {
    0x94, 0x20, 0x85, 0x10, 0xC2, 0xC0, 0x01, 0xFB,
    0x01, 0xC0, 0xC2, 0x10, 0x85, 0x20, 0x94, 0x01
};

```

```

};

// poly multiplication mod  $p(x) = x^8 + x^7 + x^6 + x + 1$ 
// totally not constant time

static uint8_t kuz_mul_gf256(uint8_t x, uint8_t y)
{
    uint8_t z;

    z = 0;
    while (y) {
        if (y & 1)
            z ^= x;
        x = (x << 1) ^ (x & 0x80 ? 0xC3 : 0x00);
        y >>= 1;
    }

    return z;
}

// linear operation l

static void kuz_l(w128_t *w)
{
    int i, j;
    uint8_t x;

    // 16 rounds
    for (j = 0; j < 16; j++) {

        // An LFSR with 16 elements from  $GF(2^8)$ 
        x = w->b[15]; // since lvec[15] = 1

        for (i = 14; i >= 0; i--) {
            w->b[i + 1] = w->b[i];
            x ^= kuz_mul_gf256(w->b[i], kuz_lvec[i]);
        }
        w->b[0] = x;

        /*lab debug*/ //printf("      L (%02d)", j);
        /*lab debug*/ //print_w128( (w128_t *) w );
    }
}

// inverse of linear operation l

static void kuz_l_inv(w128_t *w)
{
    int i, j;
    uint8_t x;

```



```

// 16 rounds
for (j = 0; j < 16; j++) {

    x = w->b[0];

    for (i = 0; i < 15; i++) {
        w->b[i] = w->b[i + 1];
        x ^= kuz_mul_gf256(w->b[i], kuz_lvec[i]);
    }
    w->b[15] = x;
}
}

// a no-op

void kuz_init()
{
    ;
    setlocale(LC_ALL, "Russian");
}

// key setup

void kuz_set_encrypt_key(kuz_key_t *kuz, const uint8_t key[32])
{
    int i, j;
    w128_t c, x, y, z;

    for (i = 0; i < 16; i++) {
        // this will be have to changed for little-endian systems
        x.b[i] = key[i];
        y.b[i] = key[i + 16];
    }

    kuz->k[0].q[0] = x.q[0];
    kuz->k[0].q[1] = x.q[1];
    kuz->k[1].q[0] = y.q[0];
    kuz->k[1].q[1] = y.q[1];

    for (i = 1; i <= 32; i++) {

        // C Value
        c.q[0] = 0;
        c.q[1] = 0;
        c.b[15] = i;          // load round in lsb
        kuz_l(&c);

        z.q[0] = x.q[0] ^ c.q[0];
        z.q[1] = x.q[1] ^ c.q[1];
        for (j = 0; j < 16; j++)
            z.b[j] = kuz_pi[z.b[j]];
        kuz_l(&z);
    }
}

```

```

        z.q[0] ^= y.q[0];

z.q[1] ^= y.q[1];

        y.q[0] = x.q[0];
        y.q[1] = x.q[1];

        x.q[0] = z.q[0];
        x.q[1] = z.q[1];

        if ((i & 7) == 0) {
            kuz->k[(i >> 2)].q[0] = x.q[0];
            kuz->k[(i >> 2)].q[1] = x.q[1];
            kuz->k[(i >> 2) + 1].q[0] = y.q[0];
            kuz->k[(i >> 2) + 1].q[1] = y.q[1];
        }
    }
}

// these two funcs are identical in the simple implementation

void kuz_set_decrypt_key(kuz_key_t *kuz, const uint8_t key[32])
{
    kuz_set_encrypt_key(kuz, key);
}

// encrypt a block - 8 bit way

void kuz_encrypt_block(kuz_key_t *key, void *blk)
{
    int i, j;
    w128_t x;

    x.q[0] = ((uint64_t *) blk)[0];
    x.q[1] = ((uint64_t *) blk)[1];

    /*lab*/ printf("Шифрование \n");
    /*lab*/ printf("Исходный текст  ");
    /*lab*/ print_w128( (w128_t *)blk );

    for (i = 0; i < 9; i++) {

        x.q[0] ^= key->k[i].q[0];
        x.q[1] ^= key->k[i].q[1];

        /*lab*/ printf("Раундовый ключ (%d)", i);
        /*lab*/ print_w128( (w128_t *) &key->k[i] );

        /*lab*/ printf("    Результат X ");
        /*lab*/ print_w128( (w128_t *) &x );
    }
}

```

```

        for (j = 0; j < 16; j++)
            x.b[j] = kuz_pi[x.b[j]];

        /*lab*/ printf("    Результат PI");
        /*lab*/ print_w128( (w128_t *) &x );

    kuz_l(&x);

    /*lab*/ printf("    Результат L ");
    /*lab*/ print_w128( (w128_t *) &x );

}
((uint64_t *) blk)[0] = x.q[0] ^ key->k[9].q[0];
((uint64_t *) blk)[1] = x.q[1] ^ key->k[9].q[1];

    /*lab*/ printf("Шифротекст: ");
    /*lab*/ print_w128( (w128_t *) &x );
    /*lab*/ printf("\n");
}

// decrypt a block - 8 bit way

void kuz_decrypt_block(kuz_key_t *key, void *blk)
{
    int i, j;
    w128_t x;

    x.q[0] = ((uint64_t *) blk)[0] ^ key->k[9].q[0];
    x.q[1] = ((uint64_t *) blk)[1] ^ key->k[9].q[1];

    for (i = 8; i >= 0; i--) {

        kuz_l_inv(&x);
        /*lab*/ printf("    Результат обратного L ");
        /*lab*/ print_w128( (w128_t *) &x );

        for (j = 0; j < 16; j++)
            x.b[j] = kuz_pi_inv[x.b[j]];

        /*lab*/ printf("    Результат обратного PI");
        /*lab*/ print_w128( (w128_t *) &x );
        x.q[0] ^= key->k[i].q[0];
        x.q[1] ^= key->k[i].q[1];

        /*lab*/ printf("Раундовый ключ (%d)", i);
        /*lab*/ print_w128( (w128_t *) &key->k[i] );
    }
}

```

```
        /*lab*/ printf("    Результат X ");
        /*lab*/ print_w128( (w128_t *) &x );
    }
    ((uint64_t *) blk)[0] = x.q[0];
    ((uint64_t *) blk)[1] = x.q[1];

    /*lab*/ //print_w128( (w128_t *) &x );
    /*lab*/ printf("\n");
}

// debug print state

void print_w128(w128_t *x)
{
    int i;

    for (i = 0; i < 16; i++)
        printf(" %02X", x->b[i]);
    printf("\n");
}
```