ДОДАТОК А
ВИКОРИСТОВУВАНІ ТАБЛИЦІ ПІДСТАНОВОК

Випадкова таблиця підстановки:

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

Стандартна таблиця підстановки:

Sbox_Standart[256] = {
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};

ДОДАТОК Б
С++ КОД БСШ «ШУП»

```
#include <iostream>
#include <time.h>
#include <fstream>
#include <stdlib.h>

using namespace std;

#define u8 unsigned __int8
#define u32 unsigned __int32
#define u64 unsigned __int64

// случайный Sbox
u8 Sbox_Random[256] = {
0x6A, 0xF5, 0xC2, 0x2B, 0x63, 0x21, 0x3F, 0x47, 0x8C, 0x38, 0xE8, 0x42, 0x60, 0x79, 0xCE,
0xA1,
0xA9, 0x2E, 0xFD, 0xD5, 0x6E, 0x1A, 0x14, 0xFC, 0x0E, 0xE6, 0x22, 0x96, 0x8A, 0xA7, 0xF1,
0x8E,
0xE9, 0xB1, 0xB4, 0x30, 0x0F, 0x5E, 0xB6, 0xC9, 0xC4, 0xD7, 0x18, 0xD6, 0x09, 0x1F, 0x57,
0x93,
0xE3, 0x3D, 0xCF, 0xD2, 0x6D, 0x76, 0x0D, 0x83, 0xE5, 0x7A, 0x29, 0xEA, 0x43, 0xFB, 0x16,
0x9A,
0xCB, 0xF4, 0xA6, 0xE1, 0x32, 0x5A, 0xC7, 0x65, 0x9D, 0xD3, 0x37, 0x9C, 0x95, 0xB9, 0x34,
0x07,
0x11, 0x19, 0x71, 0x4E, 0x3B, 0xD9, 0x90, 0x03, 0x4A, 0x1D, 0x05, 0x99, 0x6B, 0x39, 0x3A,
0x80,
0x81, 0xB2, 0x40, 0x8D, 0x8B, 0x01, 0xE4, 0xFE, 0x91, 0xEB, 0xA0, 0xDD, 0xB5, 0xC6, 0xB8,
0x31,
0x45, 0xE2, 0x70, 0x41, 0xC5, 0xCA, 0x24, 0x15, 0xEE, 0x4D, 0x28, 0xBB, 0x9B, 0x87, 0x7C,
0xDF,
0x56, 0xF3, 0x7E, 0xAB, 0x52, 0xE7, 0x94, 0x20, 0xFF, 0x1C, 0x00, 0x02, 0x3E, 0x1B, 0x51,
0x67,
0x0A, 0x55, 0xA5, 0x33, 0x85, 0x49, 0x89, 0x7F, 0x48, 0xAA, 0xF6, 0xAC, 0xDE, 0x92, 0x5C,
0x23,
0xDC, 0xA4, 0x9F, 0xBF, 0x61, 0x5D, 0x12, 0xBD, 0x35, 0xFA, 0xBE, 0xF9, 0xD8, 0x72, 0xCC,
0x25,
0x27, 0x69, 0xDB, 0x6F, 0x4C, 0x59, 0x3C, 0x54, 0xBA, 0x75, 0x2D, 0xD1, 0x0B, 0x9E, 0x82,
0x8F,
0x77, 0x86, 0xA3, 0xA2, 0x66, 0x73, 0xEC, 0xAE, 0x36, 0xC1, 0xD4, 0x2F, 0x88, 0x46, 0x1E,
0x2A,
0xF0, 0xEF, 0x06, 0x5B, 0xB7, 0xA8, 0xC3, 0xB3, 0x13, 0x08, 0x7D, 0x10, 0x68, 0xF8, 0xAF,
0x98,
0x0C, 0x84, 0xBC, 0x4F, 0x78, 0x04, 0x6C, 0x64, 0x62, 0xD0, 0xCD, 0xF2, 0xC8, 0x50, 0xDA,
0x44,
0xC0, 0xB0, 0xF7, 0x58, 0xE0, 0xAD, 0x5F, 0xED, 0x2C, 0x26, 0x53, 0x97, 0x7B, 0x4B, 0x74,
0x17};


// стандартный Sbox
u8 Sbox_Standart[256] = {
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};

u32 res[8][65536];
u8 results_in[8][32][65536];      // значения на входе SL-блока (после

u8 Multiply(u8, u8);              // функция перемножения
void SL(u32&);                    // SL-преобразование (SubBytes + MDR)
void gen_key_256();               // генерация 256-битового ключа
void text_add_key(const u32*, const u32*, u32*, bool);// сложение ключа с текстом
void round(u32*);                         // прохождения одного цикла зашифрования (Sbox)
void u8_to_u32(const u8*, u32&);          // 8-битовые значения в 32-битовые
void u32_to_u8(const u32, u8*);           // 32-битовые значения в 8-битовые
void rebuild(u32*);                       // надстройка
void char_to_u32(char*, u32*);            // преобразование считанного из файла текста (256
бит) в восемь 32-битовых значений
void encryption(bool);// зашифрование (считывание ключа из файла, шифрование) - через Sbox
void dif();                               // подсчитывание дифференциальных максимумов
bool heming(u32);                         // подсчёт веса Хеминга
void lat();                               // подсчитывание максимумов линейной аппроксимации
void lat_new();

int main()
{
	gen_key_256();
	encryption(0);
	dif();
	lat();
	encryption(1);
	dif();
	lat();
	system("pause");
	return 0;
}

// выполянет умножение в поле Галуа
u8 Multiply(u8 a, u8 b)                                 // перемножение
{
```

```
int tmp = b;
int tmp1 = b;
int tmp2 = b;
u32 mod = 0x11B;                    // mod = x^8 + x^4 + x^3 + x + 1
if (a == 0x1)                       // умножение на 1 ничего не меняет
       return (tmp);
if (a == 0x2)                       // умножение на 2 = сдвиг влево на единицу
{
       tmp = tmp << 1;
       if (tmp >= 0x100)
              tmp ^= mod;
       return ((tmp & 0xff));
}
if (a == 0x3)                       // умножение на 3 = (умножение на 2) + b
{
       tmp = tmp << 1;
       if (tmp >= 0x100)
              tmp ^= mod;
       return (((tmp ^ b) & 0xff));
}
if (a == 0x9)                       // умножение на 9 = (2 * 2 * 2) + b = 8 + b
{
       for (int i = 0; i < 3; i++)
       {
              tmp = tmp << 1;
              if (tmp >= 0x100)
                     tmp ^= mod;
       }
       return (((tmp ^ b) & 0xff));
}
if (a == 0xB)               // умножение на 11 = (2 * 2 * 2) + 2 + b = 8 + 2 + b
{
       for (int i = 0; i < 3; i++)
       {
              tmp = tmp << 1;
              if (tmp >= 0x100)
                     tmp ^= mod;
       }
       tmp1 = tmp1 << 1;
       if (tmp1 >= 0x100)
              tmp1 ^= mod;
       return (((tmp ^ tmp1 ^ b) & 0xff));
}
if (a == 0xD)               // умножение на 13 = (2 * 2 * 2) + (2 * 2) + b = 8 + 4 + b
{
       for (int i = 0; i < 3; i++)
       {
              tmp = tmp << 1;
              if (tmp >= 0x100)
                     tmp ^= mod;
       }
       for (int i = 0; i < 2; i++)
       {
              tmp1 = tmp1 << 1;
              if (tmp1 >= 0x100)
                     tmp1 ^= mod;
       }
       return (((tmp ^ tmp1 ^ b) & 0xff));
}
if (a == 0xE)               // умножение на 14 = (2 * 2 * 2) + (2 * 2) + 2 = 8 + 4 + 2
{
       for (int i = 0; i < 3; i++)
       {
              tmp = tmp << 1;
```

```
                if (tmp >= 0x100)
                        tmp ^= mod;
                }
                for (int i = 0; i < 2; i++)
                {
                        tmp1 = tmp1 << 1;
                        if (tmp1 >= 0x100)
                                tmp1 ^= mod;
                }
                tmp2 = tmp2 << 1;
                if (tmp2 >= 0x100)
                        tmp2 ^= mod;
                return (((tmp ^ tmp1 ^ tmp2) & 0xff));
        }
}

// SL-преобразование на основе Sbox-а
void SL(u32& state)                     // SybBytes + MixColumns
{
        u8 state_u8[4], tmp[4];
        u32_to_u8(state, state_u8);     // разбивание 32-битового слова на четыре байта
        for(int i = 0; i < 4; i++)
        {
                state_u8[i] = Sbox[state_u8[i]];         // SubBytes
                results_in[global_Nround][global_Nbyte + i][global_Ntext] = state_u8[i];
        }
        tmp[0] = Multiply(0x2, state_u8[0]) ^ Multiply(0x3, state_u8[1]) ^ state_u8[2] ^
state_u8[3];
        tmp[1] = state_u8[0] ^  Multiply(0x2, state_u8[1]) ^ Multiply(0x3, state_u8[2]) ^
state_u8[3];
        tmp[2] = state_u8[0] ^  state_u8[1] ^ Multiply(0x2, state_u8[2]) ^ Multiply(0x3,
state_u8[3]);
        tmp[3] = Multiply(0x3, state_u8[0]) ^ state_u8[1] ^ state_u8[2]  ^ Multiply(0x2,
state_u8[3]);
        for(int i = 0; i < 4; i++)
                state_u8[i] = tmp[i];// запись результатов перемножения в байты состояния
        u8_to_u32(state_u8, state);// четыре байта в 32-битовое слово
}
// генерирует случайный 256-битовый ключ
void gen_key_256()                // генерация ключа (ничего интересного)
{
        srand(time(NULL));
        ofstream out_key;
        u32 key_32;
        out_key.open("key.txt");
        for(int i = 0; i < 8; i++)
        {
                key_32 = 1;
                for(int j = 0; j < 31; j ++)
                {
                        key_32 <<= 1;
                        key_32 ^= rand() % 2;
                }
                out_key << hex << key_32;
        }
        out_key.close();
}

// складывает текст и ключ по модулю два или 2^32
void text_add_key(const u32* text, const u32* key, u32* state, bool trig)  // trig = 0 -
XOR, trig = 1 - mod 2^32
{
        if(trig == 0)
                for(int i = 0; i < 8; i++)
```

```
                state[i] = text[i] ^ key[i];
        if(trig == 1)
        {
                u64 temp;
                for(int i = 0; i < 8; i++)
                {
                        temp = (text[i] + key[i]) % 4294967296;         // 2^32 = 4294967296
                        state[i] = temp;
                }
        }
}

// выполняет цикловое преобразование (Sbox)
void round(u32* state)                                  // state - блок состояния
{
        global_Nbyte = 0;
        SL(state[0]);
        for(int i = 1; i < 8; i++)                      // каждое предыдущее состояние
ксорится с текущим состоянием, после чего выполняется SL-преобразование
        {
                state[i] ^= state[i-1];
                global_Nbyte = i * 4;
                SL(state[i]);
        }
        state[0] ^= state[7];// результаты восьмого SL-преобразования ксорятся с результатами
первого
}

// переводит байтовый массив из 4 значений в 32-битовой слово
void u8_to_u32(const u8* state_u8, u32& state_u32)
{
        state_u32 = state_u8[0];
        for(int i = 1; i < 4; i++)
        {
                state_u32 <<= 8;
                state_u32 ^= state_u8[i];
        }

}

// переводит 32-битовое слово в байтовый массив из 4 значений
void u32_to_u8(const u32 state_u32, u8* state_u8)
{
        for(int i = 0; i < 4; i++)
                state_u8[i] = (state_u32 >> (24 - (i * 8)));
}

// осуществялет надстройку
void rebuild(u32* text)                 // надстройка вида
x[0]=x[0]^x[1]^x[2]^x[3]^x[4]^x[5]^x[6]^x[7],      x[1]=x[1]^x[3]^x[5]^x[7],
x[2]=x[2]^x[6],       x[3]=x[3]^x[7]
{                                                       // x[4-7] остаются без изменения
        text[0] = (text[0] ^ text[1] ^ text[2] ^ text[3] ^ text[4] ^ text[5] ^ text[6] ^
text[7]);
        text[1] = (text[1] ^ text[3] ^ text[5] ^ text[7]);
        text[2] = (text[2] ^ text[6]);
        text[3] = (text[3] ^ text[7]);
}

// снимает надстройку
void rebuild_inv(u32* text)       // надстройка в обратном порядке
{
        text[3] = (text[3] ^ text[7]);
        text[2] = (text[2] ^ text[6]);
```

```
	text[1] = (text[1] ^ text[3] ^ text[5] ^ text[7]);
	text[0] = (text[0] ^ text[1] ^ text[2] ^ text[3] ^ text[4] ^ text[5] ^ text[6] ^
text[7]);
}

// текст считывается из файла в десятичной форме, хотя записывается в 16-ричной. Приводит в
норму значение.
void char_to_u32(char* key_char, u32* key_u32)		// key_char - считанный из файла
текст, key_u32 - 32-байтовые слова
{
	u8 key_u8[32];                                                              // вся
суть функции в том, что в файл записываются ключ/текст в шестнадцатиричной системе, а
считывается - в десятичной,
	for(int i = 0, j = 0; i < 32; i++, j += 2)		// поэтому необходимо
преобразовать текст в необходимый вид
	{
		if((key_char[j] >= 'a') && (key_char[j] <= 'f'))
			key_char[j] -= 87;
		else
			key_char[j] -= 48;
		if((key_char[j+1] >= 'a') && (key_char[j+1] <= 'f'))
			key_char[j+1] -= 87;
		else
			key_char[j+1] -= 48;
		key_u8[i] = key_char[j] * 16 + key_char[j+1];
	}
	int k = 0;
	for(int i = 0; i < 8; i++)
	{
		key_u32[i] = 0;
		for(int j = 0; j < 4; j++)
		{
			key_u32[i] <<= 8;
			key_u32[i] ^= key_u8[k];
			k++;
		}
	}
}

// шифрование с обычным Sbox-ом
void encryption(bool trig)
{
	ifstream in_key;
	char key_u8[65] = {0};
	u32 text_u32[8];
	u32 key_u32[8];
	u32 state[8];
	in_key.open("key.txt", ios_base::in|ios_base::binary);
	in_key >> key_u8;                                       // считавание ключа
	char_to_u32(key_u8, key_u32);                   // перобразование ключа в чисельный
вид
	in_key.close();
	for(int Ntext = 0; Ntext < 65536; Ntext++)
	{
		global_Ntext = Ntext;
		//for(int i = 1; i < 8; i++)
		//	text_u32[i] = 0;
		//text_u32[0] = Ntext;

		for(int i = 0; i < 7; i++)
			text_u32[i] = 0;
		text_u32[7] = Ntext;
		if(trig == 1)
			rebuild(text_u32);
```

```
		for(int i = 0; i < 8; i++)
			state[i] = text_u32[i];
		for(int Nround = 0; Nround < 8; Nround++)
		{
			global_Nround = Nround;
			text_add_key(state, key_u32, state, 0);		// сложение ключа с
текстом
			round(state);							//
раундовое преобразование
			res[Nround][Ntext] = (state[0] >> 16);		// считывание
результатов на каждом цикле
			//res[Nround][Ntext] = (state[7] & 0xFFFF);
		}
	}
}

// вычисляет DDT и записывает в файл max_dif0.txt
void dif()
{
	ofstream out;
	u32 max[8];
	u32 diff_table[65536];
	memset(max, 0, sizeof(max));
	for(int Nround = 0; Nround < 4; Nround++)
		for(int dX = 1; dX < 65536; dX++, system("cls"), printf("DDT: Round %d:
%.2f%%", Nround + 1, ((double)dX/(double)65536)*(double)100))
		{
			memset(diff_table, 0, sizeof(diff_table));
			for(int Ntext = 1; Ntext < 65536; Ntext++)
				diff_table[res[Nround][Ntext]^res[Nround][dX^Ntext]]++;
			for(u64 Ntext = 0; Ntext < 65536; Ntext++)
				if(diff_table[Ntext] > max[Nround])
					max[Nround] = diff_table[Ntext];		// поиск
максимума на каждом раунде
		}
	out.open("max_DDT.txt");
	for(int Nround = 0; Nround < 4; Nround++)
		out << Nround+1 << ". " << max[Nround] << endl;
	out.close();
}

// вес Хэминга
bool heming(u32 value)
{
	bool mask = true;
	bool weight = false;
	for(int i = 0; i < 16; i++){
		weight ^= ((value >> i) & mask);
	}
	return weight;
}

void lat(){
	ofstream out;
	u32 max[8];
	u32 lat[65536];
	u32 temp = 0;
	u32 NColumns = 65536;
	u32 NRow = 65536;
	u32 N = 2;
	memset(max, 0, sizeof(max));
	for(int Nround = 0; Nround < 4; Nround++)
	{
```

```
		for(int countColumns = 1; countColumns < NColumns; countColumns++,
system("cls"), printf("LAT: Round %d: %.2f", Nround + 1,
((double)countColumns/(double)65535)*(double)100))
		{
			for(int countRow = 0; countRow < NRow; countRow++)
				lat[countRow] = heming(res[Nround][countRow] & countColumns);
			for(int countRow = 0; countRow < NRow; countRow += 2)
			{
				if((lat[countRow] == 0) && (lat[countRow + 1] == 0))
				{
					lat[countRow] = 1;
					lat[countRow + 1] = 0;
					continue;
				}
				if((lat[countRow] == 0) && (lat[countRow + 1] == 1))
				{
					lat[countRow] = 0;
					lat[countRow + 1] = 1;
					continue;
				}
				if((lat[countRow] == 1) && (lat[countRow + 1] == 0))
				{
					lat[countRow] = 0;
					lat[countRow + 1] = -1;
					continue;
				}
				if((lat[countRow] == 1) && (lat[countRow + 1] == 1))
				{
					lat[countRow] = -1;
					lat[countRow + 1] = 0;
				}
			}
			N = 2;
			do{
				for(int countRow = 0; countRow < (NRow - N*2 + 1); countRow +=
N*2)
				{
					for(int K = 0; K < N; K++)
					{
						temp = lat[countRow + K];
						lat[countRow + K] += lat[countRow + N + K];
						lat[countRow + N + K] = temp - lat[countRow + N +
K];
					}
				}
				N *= 2;
			}while(N != NRow);
			for(int countRow = 0; countRow < NRow; countRow++)
				if(labs(lat[countRow]) > max[Nround])
					max[Nround] = labs(lat[countRow]);                    //
поиск максимума на каждом раунде
		}
	}
	out.open("max_LAT.txt");
	for(int Nround = 0; Nround < 4; Nround++)
		out << max[Nround] << endl;
	out.close();
}
// вычисляет DDT и записывает в файл max_dif1.txt
void dif()
{
	ofstream out;
	u32 max[8];
	u32 diff_table[65536];
```

```
	memset(max, 0, sizeof(max));
	for(int Nround = 0; Nround < 4; Nround++)
		for(int dX = 1; dX < 65536; dX++, system("cls"), printf("DDT: Round %d:
%.2f%%", Nround + 1, ((double)dX/(double)65536)*(double)100))
		{
			memset(diff_table, 0, sizeof(diff_table));
			for(int Ntext = 1; Ntext < 65536; Ntext++)
				diff_table[res[Nround][Ntext]^res[Nround][dX^Ntext]]++;
			for(u64 Ntext = 0; Ntext < 65536; Ntext++)
				if(diff_table[Ntext] > max[Nround])
					max[Nround] = diff_table[Ntext];		// поиск
максимума на каждом раунде
		}
	out.open("max_DDT1.txt");
	for(int Nround = 0; Nround < 4; Nround++)
		out << Nround+1 << ". " << max[Nround] << endl;
	out.close();
}

void lat(){
	ofstream out;
	u32 max[8];
	u32 lat[65536];
	u32 temp = 0;
	u32 NColumns = 65536;
	u32 NRow = 65536;
	u32 N = 2;
	memset(max, 0, sizeof(max));
	for(int Nround = 0; Nround < 4; Nround++)
	{

		for(int countColumns = 1; countColumns < NColumns; countColumns++,
system("cls"), printf("LAT: Round %d: %.2f", Nround + 1,
((double)countColumns/(double)65535)*(double)100))
		{
			for(int countRow = 0; countRow < NRow; countRow++)
				lat[countRow] = heming(res[Nround][countRow] & countColumns);
			for(int countRow = 0; countRow < NRow; countRow += 2)
			{
				if((lat[countRow] == 0) && (lat[countRow + 1] == 0))
				{
					lat[countRow] = 1;
					lat[countRow + 1] = 0;
					continue;
				}
				if((lat[countRow] == 0) && (lat[countRow + 1] == 1))
				{
					lat[countRow] = 0;
					lat[countRow + 1] = 1;
					continue;
				}
				if((lat[countRow] == 1) && (lat[countRow + 1] == 0))
				{
					lat[countRow] = 0;
					lat[countRow + 1] = -1;
					continue;
				}
				if((lat[countRow] == 1) && (lat[countRow + 1] == 1))
				{
					lat[countRow] = -1;
					lat[countRow + 1] = 0;
				}
			}
			N = 2;
```

```
                    do{
                          for(int countRow = 0; countRow < (NRow - N*2 + 1); countRow +=
N*2)
                          {
                                for(int K = 0; K < N; K++)
                                {
                                      temp = lat[countRow + K];
                                      lat[countRow + K] += lat[countRow + N + K];
                                      lat[countRow + N + K] = temp - lat[countRow + N +
K];
                                }
                          }
                          N *= 2;
                    }while(N != NRow);
                    for(int countRow = 0; countRow < NRow; countRow++)
                          if(labs(lat[countRow]) > max[Nround])
                                max[Nround] = labs(lat[countRow]);                    //
поиск максимума на каждом раунде
             }
      }
      out.open("max_LAT1.txt");
      for(int Nround = 0; Nround < 4; Nround++)
             out << max[Nround] << endl;
      out.close();
}
// SL-преобразование на основе латинских квадратов
void SL_latin_square(u32& state, u32& manage_byte)                  // Latin_square +
MixColumns
{
      u8 state_u8[4];
      u32_to_u8(state, state_u8);                              // разбивание 32-битового слова
на четыре байта
      for(int i = 0; i < 4; i++)
      {
             //if((manage_byte >= 0) && (manage_byte <= 3) || (manage_byte >= 252) &&
(manage_byte <= 255))
             //     manage_byte = (manage_byte + 4) % 256;
             state_u8[i] = Sbox[(manage_byte + state_u8[i]) % 256];       // latin_square
             results_in[global_Nround][global_Nbyte + i][global_Ntext] = state_u8[i]; //
запись изменённых байтов для подсчёта активированных
             manage_byte = state_u8[i]; // запись управляющего бита
      }
      u8_to_u32(state_u8, state);                              // четыре байта в 32-битовое
слово
}

// выполняет цикловое преобразование (латинский квадрат)
void round_latin_square(u32* state, u32& manage_byte)                                 //
state - блок состояния
{
      global_Nbyte = 0;
      SL_latin_square(state[0], manage_byte);
      for(int i = 1; i < 8; i++)                        // каждое предыдущее состояние
ксорится с текущим состоянием, после чего выполняется SL-преобразование
      {
             state[i] ^= state[i-1];
             global_Nbyte = i * 4;
             SL_latin_square(state[i], manage_byte);
      }
      state[0] ^= state[7];                                    // результаты восьмого SL-
преобразования ксорятся с результатами первого
}

// шифрование с латинским квадратом
```

```
void encryption_latin_square(bool trig)
{
	ifstream in_key;
	char key_u8[65] = {0};
	u32 text_u32[8];
	u32 key_u32[8];
	u32 state[8];
	u32 manage_byte;
	in_key.open("key.txt", ios_base::in|ios_base::binary);
	in_key >> key_u8;                                        // считавание ключа
	char_to_u32(key_u8, key_u32);                    // перобразование ключа в чисельный
вид
	in_key.close();
	for(int Ntext = 0; Ntext < 65536; Ntext++)
	{
		manage_byte = 0;
		global_Ntext = Ntext;
		for(int i = 0; i < 7; i++)
			text_u32[i] = 0;
		text_u32[7] = Ntext;
		if(trig == 1)
			rebuild(text_u32);
		for(int i = 0; i < 8; i++)
			state[i] = text_u32[i];
		for(int Nround = 0; Nround < 8; Nround++)
		{
			global_Nround = Nround;
			text_add_key(state, key_u32, state, 0);         // сложение ключа с
текстом
			round_latin_square(state, manage_byte);         // раундовое
преобразование
			res[Nround][Ntext] = (state[0] >> 16);          // считывание
результатов на каждом цикле
		}
	}
}

// генерирует обратную таблицу подстановки
void gen_Sbox_inv()
{
	for(int j = 0; j < 256; j++)
		Sbox_inv[Sbox[j]] = j;
}

// обратное SL-преобразование на основе Sbox-а
void SL_inv(u32& state)                                          // SubBytes_inv +
MixColums_inv
{
	u8 state_u8[4], tmp[4];
	u32_to_u8(state, state_u8);                              // разбивание 32-битового слова
на четыре байта
	tmp[0] = Multiply(0xE, state_u8[0]) ^ Multiply(0xB, state_u8[1]) ^ Multiply(0xD,
state_u8[2]) ^ Multiply(0x9, state_u8[3]);
	tmp[1] = Multiply(0x9, state_u8[0]) ^ Multiply(0xE, state_u8[1]) ^ Multiply(0xB,
state_u8[2]) ^ Multiply(0xD, state_u8[3]);
	tmp[2] = Multiply(0xD, state_u8[0]) ^ Multiply(0x9, state_u8[1]) ^ Multiply(0xE,
state_u8[2]) ^ Multiply(0xB, state_u8[3]);
	tmp[3] = Multiply(0xB, state_u8[0]) ^ Multiply(0xD, state_u8[1]) ^ Multiply(0x9,
state_u8[2]) ^ Multiply(0xE, state_u8[3]);
	for(int i = 0; i < 4; i++)
		state_u8[i] = tmp[i];                                // запись результатов
перемножения в байты состояния
	for(int i = 0; i < 4; i++)
		state_u8[i] = Sbox_inv[state_u8[i]];     // SubBytes_inv
```

```
	u8_to_u32(state_u8, state);				// четыре байта в 32-битовое слово
}

// выполняет обратное цикловое преобразование (Sbox)
void round_inv(u32* state)				// state - блок состояния
{
	state[0] ^= state[7];					// результаты восьмого SL-преобразования ксорятся с результатами первого
	SL_inv(state[7]);
	for(int i = 6; i >= 0; i--)			// каждое предыдущее состояние ксорится с текущим состоянием, после чего выполняется SL-преобразование
	{
		state[i+1] ^= state[i];
		SL_inv(state[i]);
	}
}

// снимает надстройку
void rebuild_inv(u32* text)	// надстройка в обратном порядке
{
	text[3] = (text[3] ^ text[7]);
	text[2] = (text[2] ^ text[6]);
	text[1] = (text[1] ^ text[3] ^ text[5] ^ text[7]);
	text[0] = (text[0] ^ text[1] ^ text[2] ^ text[3] ^ text[4] ^ text[5] ^ text[6] ^ text[7]);
}

// дешифрование с обычным Sbox-ом (нужен усключительно для проверки работоспособности всего алгоритма)
void decryption(bool trig)
{
	ifstream in_key;
	char key_u8[65] = {0};
	u32 key_u32[8];
	u32 state[8];
	in_key.open("key.txt", ios_base::in|ios_base::binary);
	in_key >> key_u8;					// считавание ключа
	char_to_u32(key_u8, key_u32);		// перобразование ключа в чисельный вид
	in_key.close();
	for(int i = 0; i < 8; i++)
		state[i] = test[i];
	for(int Nround = 0; Nround < 4; Nround++)
	{
		round_inv(state);					// раундовое преобразование
		text_add_key(state, key_u32, state, 0);  // сложение ключа с текстом
	}
	if(trig == 1)
		rebuild_inv(state);
	for(int i = 0; i < 8; i++)
		test[i] = state[i];
}
```

ДОДАТОК В
ПУБЛІКАЦІЇ РЕЗУЛЬТАТІВ РОБОТИ

А) Прикладна радіоелектроніка. 2019 (в друці).
О МИНМАЛЬНОМ ЧИСЛЕ S-БЛОКОВ, АКТИВИЗИРУЕМЫХ НА ПЕРВЫХ ЦИКЛАХ SPN ШИФРОВ

ЛИСИЦКАЯ И.В. ВАСИЛЬЕВ В.А.

На примере уменьшенной модели шифра Rijndael приводятся реальные результаты определения динамических показателей прихода шифров с различными конструкциями дополнительных линейных смешивающих преобразований на их входах к состоянию случайной подстановки. Показывается, что эти дополнительные преобразования являются не эффективными для улучшения показателей случайности шифров Rijndael, ADE Калина, Кузнечик и ряда других. Подтверждается общее положение, что для всех известных SPN шифров с однослойными подстановочными цикловыми преобразованиями минимальное число активизируемых S-блоков первого цикла равно одному.
Ключевые слова: активные S-блоки, динамические показатели прихода шифра к состоянию случайной подстановки, дополнительное смешивающее линейное преобразование, распределение полных дифференциалов, распределение максимумов смещений ЛАТ, мини-версия шифра Rijndael, шифр с управляемыми подстановками.

ВВЕДЕНИЕ

Мы уже давно интересуемся активизацией S-блоков первого цикла блочных симметричных шифров [1-6 и др.] и изучаем возможности увеличения их минимального числа. Были рассмотрены разные схемы линейных доцикловых перемешиваний блоков данных. Однако, как теперь стало понятно, показатели случайности преобразований непосредственно связаны с использованием в них механизмов случайного перемешивания сегментов данных, участвующих в преобразованиях. Они могут быть реализованы с помощью нелинейных (подстановочных) преобразований (перемешиваний с помощью S-блоков), позволяющих обеспечить статистическую независимость сегментов на выходах преобразования. В этой работе мы приводим результаты оценки дифференциальных показателей мини версии шифра Rijndael с различными конструкциями линейных смешивающих преобразований на его входе и сравниваем полученные результаты с показателями оригинальной разработки. Приводятся также результаты экспериментов с полномасштабными 256-ти битными шифрами

1. СОСТОЯНИЕ ВОПРОСА

Ранее проведенные исследования [1-3] показали, что динамические показатели прихода шифров к состоянию случайной обстановки существенно зависят от минимального числа S-блоков, которые активизируются их первых циклах. Здесь под динамическими показателями прихода шифра к состоянию случайной подстановки понимается минимальное число циклов шифрования, после которого максимальные значения дифференциальных и линейных вероятностей принимают асимптотические значения свойственные случайным подстановкам соответствующей степени. Это и вызвало интерес к изучению возможностей увеличения числа активизируемых S-блоков на первых циклах шифрующих преобразований. В наших работах [4-6] рассматривались возможности увеличения числа активизируемых S-блоков на первых циклах шифров за счёт использования на их входах дополнительных смешивающих линейных преобразований. Было показано, что действительно дополнительные смешивающие преобразования позволяют решить задачу увеличения числа активизируемых S-блоков. Однако, как стало сейчас понятно, линейные преобразования на входах шифров не в состоянии улучшить их показателей случайности. При линейных преобразованиях нарушается (отсутствует) случайная связь S-блоками, и поэтому результирующая вероятность прохода первого цикла определяется не произведением вероятностей переходов S-блоков, как это осуществляется в цикловых преобразованиях большинства шифров, а скорее их суммой. Цель этой работы состоит в том, чтобы показать, что дополнительные линейные преобразования на входах SPN шифров являются не эффективными для улучшения их показателей случайности.

2. РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

В качестве объекта исследований, как уже отмечено выше, сначала была выбрана уменьшенная модель шифра Rijndael, взятая из работы [7]. Это шифр с 16-ти битным размером входного блока данных и ключа. Он практически повторяет структуру преобразований оригинальной конструкции шифра. Сначала выполняется забеливание с помощью 16-ти битного ключа, а затем выполняются циклы преобразований, каждый из которых включает прохождение входного блока данных через четыре S-блока (операция оригинальной разработки SubBute). В наших мини-моделях они берутся полубайтовыми и и одинаковыми. Сами S-блоки тоже взяты из работы [7]. Отдельный S-блок задаётся строкой (10, 4, 3, 11, 8, 14, 2. 12, 5, 7, 6, 15, 0, 1, 9, 13) Этот S-блок

построен по правилам формирования байтового S-блока шифра Rijndael и имеет максимальные значения дифференциальной и линейной вероятностей равные $DP^{\pi}_{\max} = LP^{\pi}_{\max} = 2^{-2}$.

Выходы S-блоков поступают на МДР матрицу сепарабельного кода размером 4×4, т.е. активизация одного S-блока распространяется на все выходные полубайты цикла (операция MixColumn в оригинальной разработке), и поэтому последующей операции ShiftRow (как в шифре Rijndael) в малой версии шифра не требуется. Заключает цикловую функцию операция сложения по модулю 2 с цикловым (раундовым) ключом (операция AddRoundKey в оригинальной разработке).

В качестве линейных преобразований рассмотрены три конструкции:

1) Сложение по модулю 2 всех полубайтов входного 16-битного блока данных на входе первого S-блока. На остальные S-блоки поступают соответствующие полубайты входного блока данных (подобно использованного в шифре.ШУП-1 [8]):

```
d1= d1^d2^d3^d4;
d2 = d2;
d3 = d3;
d4 = d4..
```

2) Сложение полубайтов входа по три на входах S-блоков цикловой функции:

```
d1 = d1^d3^d4;
d2 = d2^d1^d4;
d3 = d3^d2^d1;
d4 = d4^d3^d1;
```

3) Применение после операции забеливания полубайтовой операции линейного смешивания MixColumn к входному блоку данных.

В первых экспериментах были вычислены распределения полных дифференциалов мини версии трёхциклового шифра Rijndael с отмеченными вариантами линейных смешивающих преобразований на его входе. Результаты экспериментов приведены в таблице 1.

Таблица 1 Распределение полных дифференциалов мини версий трёхциклового шифра Rijndael с различными конструкциями линейных смешивающих преобразований на его входе

Чистый Rijndael (без доциклового преобразования)	Доцикловое преобразование: `d1= d1^d2^d3^d4; d2 = d2; d3 = d3; d4 = d4;`	Доцикловое преобразование: `d1 = d1^d3^d4; d2 = d2^d1^d4; d3 = d3^d2^d1; d4 = d4^d3^d1;`	Доцикловое преобразование - - умножение на МДР матрицу сепарабельного кода

Ключи:
[0] = 4dab
[1] = 5097
[2] = 7a39
[3] = 694f
	Ключи:
	`[0] = 79a0`
	`[1] = 7c38`
	`[2] = 4eb8`
	`[3] = 3412`
		Ключи:
		[0] = 7b7a
		[1] = ^{442f}
		[2] = 5519
		[3] = 71af
			[0] = c33
			[1] = 47c7
			[2] = 3132
			[3] = 502b
			[4] = 1288

0=2605083584	`0=2605116431`	0=2605059693	0=2605049242
2=1302347318	`2=1302303977`	2=1302393259	2=1302406612
4=325645417	`4=325636237`	4=325629048	4=325633997
6=54293282	`6=54312219`	6=54285939	6=54278384
8=6793579	`8=6792690`	8=6793103	8=6791931
10=677534	`10=678900`	10=679611	10=680739
12=56700	`12=56937`	12=56773	12=56578
14=4086	`14=4078`	14=4081	14=4000
16=246	`16=275`	16=243	16=259
18=14	`18=16`	18=8	18=17
0	`0`	0	20=1
0	`0`	0	0

Здесь в первой колонке приведены данные для исходного уменьшенного варианта построения шифра, а в других колонках результаты трёхцикловых зашифрований при применении на входах мини версии шифров трёх вариантов линейных смешивающих преобразований.

Как следует из представленных результатов, получается, что ранее приведенные в работах [4-6] результаты являются ошибочными в том смысле, что увеличение числа активизируемых S-блоков первого цикла при линейных доцикловых преобразованиях не приводит к росту показателей случайности шифров.

Действительно, линейные преобразования на входе шифра позволяют при активизации одного байта входа увеличить число активизируемых S-блоков вплоть до максимально возможного их числа, однако, как уже было отмечено выше, за счёт линейной связи между сегментами входа вероятности прохождении S-блоков первого цикла не перемножаются, как в цикловых преобразованиях шифров, а скорее складываются. Поэтому влияние линейных преобразований на входе шифра практически обесценивается. Этот вывод можно перенести и на полномасштабные шифры: Rijndel, ADE, Калина, Кузнечик и другие близкие к ним решения, в которых результаты линейного преобразования сегментов входа поступают непосредственно на ходы S-блоков.

Вторая серия экспериментов была выполнена по построению поцикловых законов распределения

максимумов смещений ЛАТ мини версий шифра Rijndael с различными конструкциями линейных смешивающих преобразований на его входе. Эти результаты представлены в таблице 2.

Видно, что показатели случайности и в первом и во втором случае оказываются практически не зависящими от наличия на входе шифров дополнительных линейных преобразований. В таблице 3 мы представляем также пример распределения смещений таблицы ЛАТ трёхциклового мини Rijndael-я без надстроек.

Мы пришли к выводу, что механизм случайного перемешивания сегментов данных на входе шифра может быть реализован только при использовании в доцикловом преобразовании нелинейных операций с помощью, например, наборов S-блоков, как в SL преобразованиях шифров ШУП [8].

Напомним, что впервые название SL преобразование было использовано при описании шифра Лабиринт [8]. Им было названо 64-х битная конструкция, в которой сначала выполнялось разбиение 64-х битного входного блока на 8-мь байтовых сегментов. Эти сегменты параллельно проходили через 8-мь одинаковых байтовых S-блока. Выходные значения байтовых S-блоков умножались на МДР матрицу размером 8×8, и 64-х битный результат складывался по модулю 2 с уникальными 64-х битными константами разными для каждого SL преобразования.

В шифрах ШУП SL преобразованиями были названы 32-ух битные преобразования, в которых использовались 4-ре байтовых S-блока .с последующим умножением их выходов на МДР матрицу размером 4×4 (как в шифре Rijndael операция MixColumn). В дальнейшем это название было обобщено в наших работах и на 8-ми байтовые конструкции, отличающиеся от SL преобразования шифра Лабиринт отсутствием в них сложения их выходов с константами.

Таблица 2

Распределение максимумов смещений ЛАТ мини версии шифра Rijndael с различными конструкциями линейных смешивающих преобразований на его входе

Число циклов шифрования	Чистый Rijndael без доциклового преобразования	Доцикловое преобразование		

		`d1= d1^d2^d3^d4;`
		`d2 = d2;`
		`d3 = d3;`
		`d4 = d4;`
			`d1 = d1^d3^d4;`
			`d2 = d2^d1^d4;`
			`d3 = d3^d2^d1;`
			`d4 = d4^d3^d1;`
				Умножение на МДР матрицу сепарабельного кода

1	16384	16384	16384	16384
2	2048	2048	2048	2048
3	824	824	824	792

Таблица 3

Распределение смещений таблицы ЛАТ для трёхциклового мини Rijndael-я.

Значение ячейки	Число ячеек	Значение ячейки	Число ячеек	Значение ячейки	Число ячеек	Значение ячейки	Число ячеек
8	213756563	208	57201541	408	1332682	608	2634
16	212504662	216	51574071	416	1088685	616	1959
24	210438115	224	46315580	424	887082	624	1457
32	207578256	232	41430048	432	720576	632	1141
40	203969929	240	36925545	440	581855	640	799
48	199662138	248	32777513	448	468448	648	620
56	194633129	256	28984027	456	376719	656	440
64	188988294	264	25528863	464	300264	664	300
72	182827740	272	22392089	472	238809	672	230
80	176187513	280	19575688	480	189688	680	153
88	169083156	288	17043218	488	149584	688	112
96	161661895	296	14777681	496	117897	696	67
104	153982422	304	12764384	504	91946	704	48
112	146069022	312	10977543	512	72186	712	46
120	138018218	320	9407155	520	55865	720	20
128	129885617	328	8030865	528	43250	728	17
136	121795187	336	6833449	536	33370	736	9
144	113743511	344	5785834	544	25617	744	8
152	105806899	352	4885864	552	19822	752	2
160	98065364	360	4107143	560	14849	760	9
168	90504280	368	3434564	568	11303	768	5
176	83214916	376	2866000	576	8824	776	1
184	76201852	384	2379813	584	6498	784	2
192	69531104	392	1969325	592	4954	792	1
200	63196168	400	1623915	600	3527	832	1

Простые соображения показали, что и в этом случае минимальное число активизируемых S-блоков первого цикла сохраняется равным одному. Как оказалось для линейных преобразований на входе цикла всегда можно подобрать вариант инициализации входа шифра, когда активизируется минимум один S-блок. Далее для подтверждения приведенных выше соображений приводятся результаты экспериментов с 256-ти битным шифром ШУП-1 [8].

Мы здесь будем интересоваться поцикловыми законами распределения максимумов дифференциалов и смещений для случая активизации шифра 16-битными входными разностями в соответствии с методикой, изложенной в работе [8]. Результаты экспериментов иллюстрируют таблицы 4 и 5. В таблице 4 представлено поцикловое распределение максимумов дифференциалов для 256-битных шифров Rijndael, Мухомор и ШУП.

Таблица 4

Поцикловое распределение максимумов дифференциалов для 256-битных шифров при шифровании 16-битных блоков данных.

Число циклов *r*	ШифрRijndael		Шифр Мухомор	Шифр ШУП	
	Максимальное значение полного дифференциала	Среднеквадратическое отклонение	Максимальное значение полного дифференциала	Первый цикл без надстройки S-блоки шифра Мухомор	Первый цикл с надстройкой, случайные S-блоки
1	2048	0	18	8192	9728
2	3652,26	±630,312	20	20	20
3	19,0666	± 1,436	18	18	18
4	19,0666	±0,99777	20	18	18
5	18,8666	±1,23108	20	18	20
6	19,1332	±0,99106	20	20	20
7	19,2666	± 1,0934	20	20	18
8	19,1332	± 1,431394	18	20	20
9	19,0666	± 1,23648	18	18	18

Шифр Rijndael в этом эксперименте приходит к стационарному значению дифференциальной вероятности на третьем цикле (активизации двух S-блоков ему не хватает). Мухомор – это шифр с двухслойными S-блочными преобразованиями цикловой функции, и он набирает необходимое минимальное число активизируемых S-блоков уже на первом цикле. Шифр ШУП с однослойным подстановочным преобразованием приходит к состоянию случайной подстановки на втором цикле, т.е.он выигрывает у шифра Rijndael один цикл. Мы здесь снова приводим результаты для двух случаев. В первом случае шифр ШУП берётся с надстройкой в виде сложения 32-ух битных сегментов входного бока данных на входе первого SL преобразования по модулю 2 а во втором без этой надстройки. Видно, что при активизации 31-го и 32-го S-блоков при наличии надстройки число активизируемых S-блоков оказывается достаточным для прихода шифра к случайной подстановке за один цикл. Однако, как уже отмечалось ранее, линейное преобразование на входе цикла не предотвращает возможности активизации лишь одного S-блока первого цикла. Без надстройки на первом цикле активизируется один S-блок и для прихода шифра к состоянию случайной подстановки ему нужен ещё один цикл. Отметим, что те же два цикла необходимы шифру для прихода к случайной подстановке и при применении в нём случайных S-блоков.

Далее в таблице 5 представляются поцикловые законы распределение максимумов смещений таблиц ЛАТ для 256-битных шифров в режиме шифрования 16-битных блоков данных. Опять рассматриваются три шифра. Снова видно, что для прихода шифра Rijndael к состоянию случайной подстановки ему необходимо три цикла. Шифр Мухомор за счёт двухслойного подстановочного циклового преобразования становится случайно подстановкой уже на первом цикле, а ШУП с однослойным цикловым преобразованием опять приходит к стационарному значению линейной вероятности без надстройки на втором цикле.

Получается, что в общем случае увеличить число активизируемых S-блоков первого цикла SPN шифров можно только с помощью применения нелинейного доциклового преобразование, как это сделано, например, в шифре Лабиринт [9].

Другое простое решение, которое позволяет увеличить минимальное число активизируемых S-блоков первого цикла – это применение двухслойной конструкции построения .цикловой функции. Это решение также предложено в работе [8].

В обще случае можно прийти к выводу, что улучшения показателей случайности шифра можно добиться, применив на его входе цикл с двухслойным подстановочным преобразованием. Этот путь может рассматриваться как реальная возможность улучшения показателей случайности известных шифров (улучшения динамических показателей их прихода к состоянию случайной подстановки). Одновременно этот подход позволяет решить важную задачу применения в шифрах случайных S-блоков практически без предварительного их отбора. Нам

представляется, что это направление симметричного шифрования заслуживает самого совершенствования технологий блочного серьёзного внимания.

Таблица 5

Поцикловые законы распределение максимумов смещений таблиц ЛАТ для 256-битных шифров при шифровании 16-битных блоков данных.

Число циклов *r*	Шифрr Rijndael		Шифр Мухомор	Шифр ШУП	
	Максимальное значение смещения линейного корпуса	Среднеквадратическое отклонение	Максимальное значение смещения линейного корпуса	Первый цикл без надстройки S-блоки шифра Мухомор	Первый цикл с надстройкой, случайные S-блоки
1	4096	0	819	9728	846;
2	9284,27	± 657,454	825	805	836;
3	818,467	± 26,8809	828	828	837;
4	815	± 28,204	825	827	806.
5	818,5	± 18,536	828	837	830
6	815,967	± 20,18	824	814	854
7	832,1	± 33,1887	820	835	823

ВЫВОДЫ.

Подводя итоги полученным результатам и уже сложившимся убеждениям и выводам можно отметить следующее:

1. Минимальное число активизируемых S-блоков на первых циклах существенно влияет на динамические показатели прихода шифров к состоянию случайной подстановки,.

2. Смешивающие линейные преобразования любой конструкции на входах известных алгоритмов блочного симметричного шифрования: Rijndel, ADE Калина, Кузнечик и других близких к ним решений, не улучшают показателей случайности шифров, так как линейные преобразованиях не имеют механизма случайного перемешивания сегментов входного блока данных.

3. Одной из возможностей увеличения числа активизируемых S-блоков первых циклов можно считать замену первого цикла более эффективной конструкцией, построенной на основе управляемых подстановок [8]. Этот подход позволяет решить задачу доведения до максимума числа активизируемых S-блоков второго цикла.

4. Другой возможностью увеличения минимального числа активизируемых S-блоков первого цикла является применение в первом цикле двухслойного подстановочного преобразования [8].

5. Увеличение минимального числа активизируемых S-блоков первых циклов является эффективным путём построения шифров с применением случайных подстановок.

Литература

{1} Лисицкая И. В. Экспериментальные данные по определению динамических показателей прихода блочных симметричных шифров к состояни. случайной подстановки / И. В. Лисицкая, К. Е. Лисицкий, М. Ю.Родинко и др. // Радиоэлектроника, информатика, управление Запорожье: ЗНТУ – 2017. – № 1(40) – С. 129-141.

[2] Горбенко И.Д. Уточнённые показатели прихода шифров к состоянию случайной подстановки / И.Д. Горбенко, В.И. Долгов, Лисицкий К.Е. // Прикладная радиоэлектроника. – Харьков: ХНУРЭ. – 2014. Том. 13, № 3. С. 213-216.

[3] Gorbenko I.D. On Ciphers Coming to a Stationary State of Random Substitution / I.D. Gorbenko, K.E. Lisitskiy, D.S. Denisov / Universal Journal of Electrical and Electronic Engineering, 2, 206-215. doi: 10.13189/ujeee.2014.020409.

[4] Долгов В.И. Усовершенствованный блочный симметричный шифр Калина / В.И. Долгов, И.В. Лисицкая, К.Е. Лисицкий // 0485-8972. – Радиотехника Всеукр. межвед. научн.-техн. сб. – 2016. – Вип.186. – С. 119-131.

[5] Лисицкий К.Е. Усовершенствованный Rijndael / К.Е. Лисицкий, // Материалы VI Международной научно-технической конференции "Захист інформації і безпека інформаційних систем". – Львів – 1-2 червня 2017 р. С. 85-86.

[6] Lisitskaya Iryna Impruvd Rijndael / Iryna Lisitskaya, Konstantin Lisitskiy, Mariya Rodinko // Science and Education Studies "Stanford University Press" Volume II – № 1(17), January- June – 2016. p. 608-618.

[7] Долгов В.И. Методология оценки стойкости блочных симметричных шифров к атакам дифференциального и линейного криптоанализа / В.И. Долгов, И.В. Лисицкая. Монография – Харьков: Издательство ФОРТ.

[8] Dolgov V.I. The new concept of block symmetric ciphers design / V.I. Dolgov, I.V. Lisitska, K.Ye. Lisitskyi // DOI: 10.1615 /TelecomRadEng. v. 76. i. 2. pages 157-184. – 2013. – 420 c.

[9] Головашич С.А. Спецификация алгоритма блочного симметричного шифрования «Лабиринт» / С.А. Головашич // Прикладная радиоэлектроника. – Харьков: ХТУРЭ. – 2007. – Том. 6, №2. – С. 230-240..

Поступила в редакцию . .2019.

Лисицкая Ирина Викторовна, профессор ХНУ им. В.Н. Каразина. Область научных интересов: технологии блочного симметричного шифрования.

Васильев Василий Александрович, магистрант ХНУ радиоэлектроники. Область научных интересов: технологии блочного симметричного шифрования

УДК 621. 3.06

Про мінімальне число S-блоків, що активізуються на перших циклах SPN шифрів / І.В. Лисицька, В.А. Васильєв // Прикладна радіоелектроніка: научно-техн. Журнал. – 2018. – Том ХХ. – № Х. – С. ХХХ-ХХХ.

На прикладі зменшеною моделі шифру Rijndael наводяться реальні результати визначення динамічних

показників приходу шифрів з різними конструкціями додаткових лінійних змішувальних перетворень на їх входах до стану випадкової підстановки. Показується, що ці додаткові перетворення є неефективними для поліпшення показників випадковості шифрів Rijndael, ADE Калина, Коник і ряду інших. Підтверджується загальне положення, що для всіх відомих SPN шифрів з одношаровими підстановлювальними цикловими перетвореннями мінімальне число активізується S-блоків першого циклу дорівнює одному. S-блоків до 4-х.

Ключові слова: керовані підстановки, випадкові підстановки, динамічні показники приходу шифру до стану випадкової підстановки, операція введення циклових подключей, активовані S-блоки, диференціальні та лінійні показники стійкості.

Табл. 5. Іл. 0. Бібліогр. 9 назв

UDC 621. 3.06

About the minimum number of S-blocks activated on the first cycle SPN ciphers / I.V. Lisitskaya V.A. Vasiliev // Applied radio electronics: scientific and technical. Magazine. – 2019. – Volume XX. – No. X. - P. XXX-XXX.

Using the reduced model of the Rijndael cipher as an example, real results of determining the dynamic indicators of the arrival of ciphers with various designs of additional linear mixing transformations at their inputs to the state of random substitution are presented. It is shown that these additional transformations are not effective for improving the randomness indices of the ciphers Rijndael, ADE Kalina, Grasshopper and several others. The general position is confirmed that for all known SPN ciphers with single-layer permutation cyclic transformations, the minimum number of activated S-blocks of the first cycle is equal to one.

Key words: controlled substitutions, random substitutions, dynamic indicators of cipher arrival to the state of random substitutions, operation of input of loop subkeys, activated S-blocks, differential and linear stability indicators.

Tabl. 05. Fig. 0. Ref. 9 names

Б) Прикладна радіоелектроніка. 2019 (в друці).

ДОСЛІДЖЕННЯ ПОКАЗНИКІВ ВИПАДКОВОСТІ ШИФРУ ШУП

ЛИСИЦЬКА І.В., ВАСІЛЬЄВ В.А.

Вивчається механізм активізації S-блоків шифру ШУП, який розглядається як перспективне рішення по побудуванню сучасного блочного симетричного шифру. Наводяться результати експериментів по визначення впливу на показники стійкості шифру операцій введення циклових підключів. Робиться висновок, що ці операції не змінюють мінімальну кількість S-блоків, що активізуються на перших циклах зашифрування, тобто показники стійкості шифру не змінюються при різних операціях введення циклових підключів. Розглядаються диференціальні показники шифрів ШУП-1М з різними S-блоками. Обговорюється конструкція циклової функції шифру ШУП з двома шарами підстановлювальних перетворень. Аналізуються мінімальні кількості активізуємих S-блоків на перших циклах, котрі можуть бути реалізовані в конструкціях шифрів ШУП і модернізованих шифрів, побудованих з використанням керованих підстановок. Робиться висновок, що описані в роботі шифри представляються перспективними рішеннями по побудуванню сучасних SPN шифрів.

Ключові слова: керовані підстановки, випадкові підстановки, динамічні показники приходу шифру до стану випадкової підстановки, операція введення циклових підключів, активовані S-блоки, диференціальні та лінійні показники стійкості.

ВСТУП

В роботі [1] запропонована конструкція шифру ШУП з покращеними показниками приходу до стану випадкової підстановки. Основою цієї конструкції є побудування циклової функції шифру з використанням керованих підстановок (укрупнених S-блоків) названих SL перетвореннями. Вони включають четвірки S-блоків, виходи котрих піддаються множенню на МДВ матрицю розміром 4×4 (практично це операція MixColumn у шифрі Rijndael). В. основній версії шифру [1] запропонована операція введення циклових підключів за допомогою підсумування 32-ох бітних сегментів вхідного блоку даних з відповідними сегментами циклового підключу за модулем 2^{32}. В той же час в багатьох шифрах, у тому числі і у шифру Rijndael використовується для введення циклових підключів операція підсумування за модулем 2.

У цій роботі ставиться задача з'ясувати вплив операції введення циклових підключів на показники випадковості шифру ШУП.

1. ОГЛЯД ЛІТЕРАТУРИ

Сьогодні шифр Rijndael [2] рахується останнім досягненням криптографічної думки. В нашій роботі [1] наводиться аналіз останніх конструкторських рішень по побудуванню сучасних шифрів. Зокрема, в якості лідерів сучасних технологій блокового симетричного шифрування поряд з шифром Rijndael виділені шифри IDIA NXT [3], Калина [4], Мухомор [5], Кузнечик [6] та білоруський шифр [7]. Робота [1] спрямована на шифри пост-квантового періоду розвитку криптографії, і тому там цікавляться шифрами з розміром блоку шифрування та ключа, що дорівнюють або перевищують значення 256 [8]. Саме на цю довжину блоку шифрування орієнтований шифр ШУП. В [1] обгрунтовується, що шифр ШУП має переваги по відношенню до перелічених шифрів (шифр "Кузнечик" орієнтований на довжину блоку шифрування 128-м бітів). Ця перевага міститься у більш високих динамічних показниках приходу шифру до стану випадкової підстановки. У цьому шифрі на перших трьох циклах зашифрування

активізується близько 65-ти S-блоків, що дозволяє йому стати випадковою підстановкою навіть при використанні випадкових S-блоків взятих з виходу генератора випадкових підстановок практично без відбору. Це єдиний шифр з одношаровим підстановлювальним перетворенням, котрий володіє такою властивістю.

Зараз почалося інтенсивне вивчення цієї пропозиції. Вже є кілька робот, присвячених досконалому вивченню властивостей випадковості цієї конструкції і пошуку можливостей її вдосконалення [9-14]. Значна увага була приділена пошуку додатних лінійних перетворень на вході першого циклу з метою підвищення мінімального числа S-боків,що активується на першому циклі [10-12]. Однак останні дослідження в цьому напрямку показали, що цей напрямок досліджень є хибним. Лінійні перетворення на вході в шифру не в змозі поліпшити його властивостей. Збереглося загальне положення, що мінімальне число активізуємих S-блоків першого циклу SPN шифру дорівнює одному. Всебічне дослідження властивостей шифру продовжується. В цій роботі ми цікавимося ще одним питанням. Ми будемо вивчати вплив на результати шифрування операції введення циклових ключів, а також ще раз повернемося до диференціальних показників.

2. МАТЕРІАЛИ І МЕТОДИ

В цій роботі ми описуємо модернізовану конструкцію шифру ШУП, котра названа шифром ШУП-1М, де додатні підсумування сегментів вхідного блоку даних на вході SL перетворення першого циклу виключені [1]. Виключені також підсумування виходу останнього SL перетворення з виходами попередніх SL перетворень, крім складання з виходом першого SL перетворення, тому що як показали експерименти ці операції до покращення показників випадковості шифру не приводять. Схема самої конструкції наведена на Рис. 1.

Основним перетворенням циклової функції є SL перетворення. Воно повторює конструкцію SL перетворення шифру Мухомор і наведено на Рис. 2.

Останні вісім циклів повторяють конструкцію першого циклу (для реалізації конвеєрної обробки даних цикли, починаючи з четвертого, будуються без операції складання виходу останнього SL перетворення з виходами попередніх SL перетворень).

ї

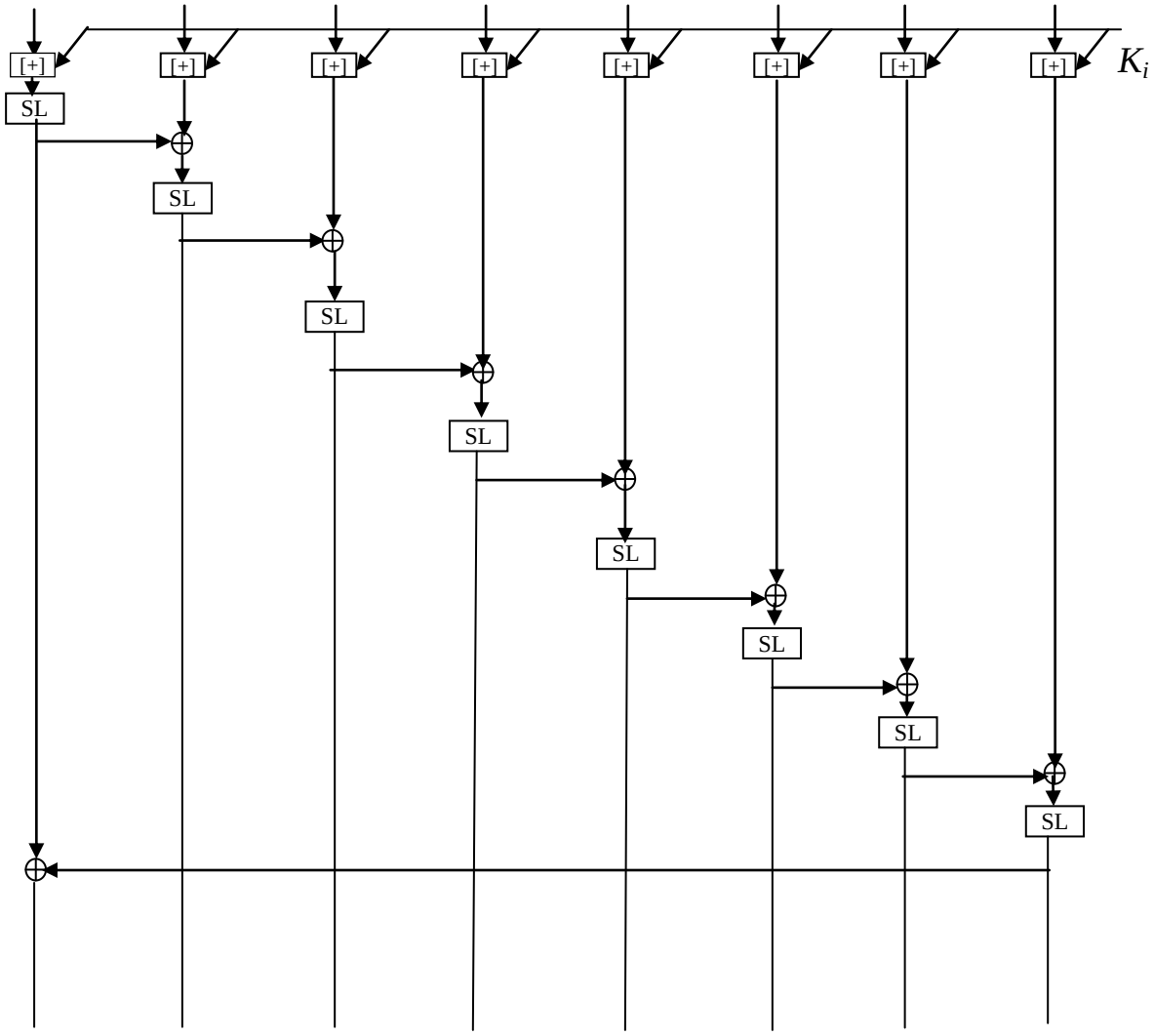

Рис. 1. Структурна схема циклового перетворення ШУП-1М

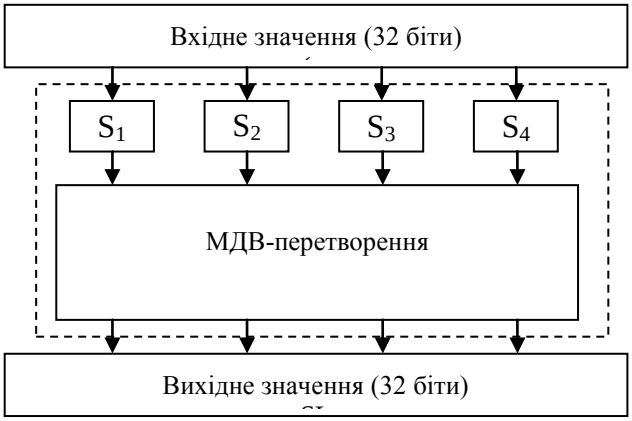

Рис. 2. – SL-перетворення

Вхідне 32-бітове значення ділиться на 4 байти, кожен з яких замінюється відповідно до заданої таблиці підстановки. У перетворенні використовується 4-ри різні таблиці, по одній на кожен байт

Після операції заміни в S-блоках 4 байти (a_0,a_1,a_2,a_3) подаються на вхід МДВ перетворення, яке виконує матричне множення наступного виду:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 02 \cdot a_0 \oplus 03 \cdot a_1 \oplus 01 \cdot a_2 \oplus 01 \cdot a_3 \\ 01 \cdot a_0 \oplus 02 \cdot a_1 \oplus 03 \cdot a_2 \oplus 01 \cdot a_3 \\ 01 \cdot a_0 \oplus 01 \cdot a_1 \oplus 02 \cdot a_2 \oplus 03 \cdot a_3 \\ 03 \cdot a_0 \oplus 01 \cdot a_1 \oplus 01 \cdot a_2 \oplus 02 \cdot a_3 \end{bmatrix}.$$

Матриця МДВ перетворення шифру Мухомор збігається з матрицею лінійного перетворення алгоритму Rijndael-AES, але при обчисленні добутку елементів вектору на матричні коефіцієнти у шифрі Мухомор використовується інший поліном

$$m(x) = x^8 + x^4 + x^3 + x^2 + 1,$$

або {01}{1d} в шістнадцятковому поданні.

Вихідний 32-бітний вектор МДВ перетворення (b_0,b_1,b_2,b_3) є вихідним значенням L перетворення.

3. ЕКСПЕРИМЕНТИ

В першому нашому експерименті ми розглянули шифр ШУП-1М з операцією введення циклового ключа як в оригінальної пропозиції з допомогою операції підсумування за модулем 2^{32}. Результати експерименту ілюструє таблиця 1. В експерименті на вхід шифру подавалися 16-ти бітні блоки даних. Активізувалися 31-ий і 32-ий S-блоки першого циклу. На виході шифру також використовувалися 16-ти бітні вихідні блоки (різниці), тобто великі шифри запускалися як малі, відповідно до методики з роботи [9]. Нагадаємо, що активним S-блоком називається S-блок з ненульовою вхідною чи вихідною різницею.

Таблиця 1

Поциклові закони розподілу числа активованих S-блоків при підсумуванні циклових підключів за модулем 2^{32}

Кількість активованих S-блоків	Перший цикл		Другий цикл		Третій цикл
	Загальне число переходів	Доля в % від загального числа переходів	Загальне число переходів	Доля в % від загального числа переходів	Загальне число переходів
1	27350220	0,63	0	0	0
2	2726313690	63,47	0	0	0
3	1541237850	35,88	0	0	0
. . .	0	0	0	0	0
26	0	0	10	0,00000002	12
27	0	0	640	0,000006	58
28	0	0	30636	0,00071	31888
29	0	0	1111744	0,021	1131942
30	0	0	28720538	0,668	28906212
31	0	0	475680994	11,08	475493694
32	0	0	3789357198	88,22	3789337254

Ми цікавилися мінімальною кількістю активованих S-блоків. Як видно з таблиці 1 при використанні операції введення циклового ключа за модулем 2^{32} активізується головним чином 2 S-блоки (більше 63%). Мається значна частина переходів, коли активізується три S-блоки (трохи більше 35%). В той же час залишається можливість активізації мінімум одного S-блоку. У всіх інших циклах практично з великою ймовірністю активізуються усі 32-а S-блоки.

У другому нашому експерименті розглянутий шифр ШУП-1М з операцією введення циклового ключа з допомогою операції підсумування за модулем 2. Результати експерименту ілюструє таблиця 2.

В цьому разі практично повторюються попередні результати крім першого циклу, на котрому активізуються більше 99% два S-блоки і менше 1% один S-блок.

Отже в обох випадках мінімальне число S-блоків, що активізуються на першому циклі, дорівнює одному, хоча таких переходів залишається дуже мало.

Можна також зробити загальний висновок, що розглянуті операції введення циклових ключів на кінцеві результати практично не впливають.

В наступній серії експериментів надаються результати дослідження поциклових законів розподілу максимумів диференціалів і максимумів зміщень Перший експеримент демонструє ефект випадкового перемішування блоків даних на першому циклі зашифрування. В якості об'єкта досліджень ми розглянули шифр ЩУП-1М. В табл. 3 і табл. 4 наводяться поциклові значення максимумів диференціалів та зміщень таблиць лінійних апроксимацій цього шифру при запуску шифру 16-бітними вхідними різницями, відповідно до методики

Таблиця 2

Поциклові закони розподілу числа активованих S-блоків при підсумуванні циклових підключів за модулем 2

Кількість активованих S-блоків	Перший цикл		Другий цикл		Третій цикл
	Загальне число переходів	Доля в % від загального числа переходів	Загальне число переходів	Доля в % від загального числа переходів	Загальне число переходів
1	33423360	0,78	0	0	0
2	4261478400	99,22	0	0	0
. . .	0	0	0	0	0
26	0	0	10	0,00000023	4
27	0	0	566	0,000013	786
28	0	0	30074	0,0007	31882
29	0	0	1097434	0,25	1133364
30	0	0	28604462	0,67	28900058
31	0	0	475991308	11,075	475519608
32	0	0	3789177906	88,2	3789316058

роботи [15]. В експерименті розглядалися два шифри. шифр з S-блоками шифру Мухмор і шифр з випадковими S-блоками.

Перший експеримент демонструє ефект випадкового перемішування блоків даних на першому циклі зашифрування. В якості об'єкта досліджень ми розглянули шифр ЩУП-1М. В табл. 4 і табл. 5 наводяться поциклові значення максимумів диференціалів та зміщень таблиць лінійних апроксимацій цього шифру при запуску шифру 16-бітними вхідними різницями, відповідно до методики з роботи [15]. В експерименті розглядалися два шифри. шифр з S-блоками шифру Мухмор і шифр з випадковими S-блоками.

Активованими були 31-й та 32-й байти входу. Отже виходить для диференціальних показників для S-блоку шифру Мухомор з $DP^{\pi}_{\max} = 2^{-5}$ (значення максимуму кількості переходів S-блоку 8), і отже максимальна кількість переходів першого циклу дорівнює 8×256 = 2048. Для випадкового S-блоку зі значенням максимальної кількості переходів S-блоку 12 виходить максимальна кількість переходів першого циклу дорівнює 12×256 = 3072.

Якщо в якості першого циклу розглядати двошарову конструкцію, як у шифрі ШУП-2 (ШУП-2М), то шифри з таким удосконаленням будуть мати на першому циклі до 65-ти активних S-блоків.

Такий результат буде зберігатися при будь-яких S-блоках (якщо результат фіксувати на 31-му та 32-му байтах виходу). Дійсно за рахунок зв'язку останнього SL перетворення з першим ці байти будуть формуватися з урахуванням усіх попередніх SL перетворень, тобто в результаті будуть працювати усі 32-а попередніх активних байти. Якщо вихідний результат формувати с першими двома байтами циклової функції, то в ньому будуть брати участь мінімум три S-блоки, тобто у будь-якому випадку шифр буду приходити до стану випадкової підстановки вже на першому циклі.

Для зміщень таблиць лінійних апроксимацій відповідно маємо результат, представлений у табл. 4. Тут розглядався перший цикл, як і в попередньому випадку з одношаровою конструкцією.

Таблиця 3

Поцикловий закон розподілу максимумів повних диференціалів для 256-битного шифру ШУП-1М при шифруванні 16-бітними блоками даних

Число циклів шифрування, *r*	Шифр ШУП-1М	
	Максимальне значення повного диференціалу S-блоками шифру Мухомор	Максимальні значення загального диференціалу для шифру з випадковими S-блоками
1	2560	3072
2	18	20
3	18	18
4	20	20
5	18	20

Таблиця 4

Поцикловий закон розподілу максимумів зміщень таблиць ЛАТ для 256-битного шифру при шифруванні 16-бітними блоками шифру ШУП-1М

Число циклів шифруванняя *r*	Шифр ШУП-1М	
	Максимальне зміщення лінійного корпусу з S-блоками шифру Мухомор	Максимальне зміщення лінійного корпусу з випадковими S-блоками
1	8192	9728
2	825	805
3	828	828
4	825	827
5	828	837

Для максимальної кількості переходів S-блоку 32 виходить максимальне значення повторень однакових різниць на виході першого циклу буде рівним 32×256 = 8192, а для випадкового S-блоку зі значенням максимуму кількості переходів S-блоку 38 одержуємо значення максимуму 38×256 = 9728. Як видно з результатів у всіх випадках на другому циклі маємо стаціонарне значення характерне для випадкової підстановки. Очевидно, якщо розглядалась би двошарова конструкція першого циклу, то і в цьому випадку ми мали б показники випадкової підстановки вже на першому циклі. В роботах [9 и др.] це рішення пропонується як удосконалена конструкція першого циклу. Для повномасштабного шифру до приходу до стану випадкової підстановки потрібен був би ще один цикл. Шифр ШУП-1М з одношаровими цикловими функціями приходить до стану випадкової підстановки за три цикли і за лінійними і за диференціальними показниками, але на відміну від шифру Rijndael поряд зі зменшенням на 1-2 цикли числа циклів приходу до стану випадкової підстановки у шифрі ШУП можуть використовуватися випадкові S-блоки.

4. ОБГОВОРЕННЯ РЕЗУЛЬТАТІВ

Перший важливий результат цієї роботи міститься в тому, що використання в шифрах операції введення ключів за модулем 2 чи за модулем 2^{32} не має помітних переваг одне від одного, так що при конструюванні шифрів можна завжди скористатися одним з простіших способів введення циклових ключів додаванням за модулем 2.

Другий важливий результат міститься в тому, що розглянуті конструкції шифрів ШУП представляють перспективні рішення по побудуванню БСШ з покращеними показниками стійкості і швидкодії. Особливу увагу представляє конструкція удосконаленого першого циклу..Відповідно до представлених результатів вона забезпечує активізацію майже половини S-блоків циклу. Такою властивістю не володіє не одна з відомих конструкцій. Це означає, що якщо використати запропоновану конструкцію першого циклу для шифру Rijndael-128, то в цьому шифрі мінімальне число S-блоків, що активізуються на перших двох циклах, буде близьким до 33-х, а за три цикли буде активізовано близько 49-ть S-блоків. Такий шифр зі стандартними S-блоками з великим запасом вже за два цикли прийде до стану випадкової підстановки і за диференціальними за лінійними показниками.

Rijndael-256 з удосконаленим першим циклом буде мати мінімальне число S-блоків, що активізуються на перших двох циклах, рівне 65-ти, а за три цикли число активованих S-блоків вже буде дорівнюватиме 97.

Калина-256 з удосконаленим першим циклом теж буде мати мінімальне число S-блоків, що активізуються на двох перших циклах, близьке до 65-ти, а на трьох 97-ми.

Кузнечик-128 з удосконаленим першим циклом буде мати мінімальне число S-блоків, що активізуються на перших двох циклах, рівне 33-ьом. Аналогічно можуть бути отримані оцінки для мінімального числа циклів зашифрування для інших шифрів.

Наведені вище результати підтверджують, що використання випадкових підстановок не приводить до перевищення мінімального числа циклів, при яких стандартні рішення забезпечують приход шифрів до стану випадкової підстановки, що свідчить, що удосконалені шифри не поступаються відомим конструкціям не по стійкості не по швидкодії. Наведені дані щодо оцінки перспектив використання випадкових S-блоків показують можливість без зниження стійкості застосовувати в удосконалених шифрах випадкові S-блоки, і, як свідчать результати експериментів, ці випадкові S-блоки можна брати безпосередньо з виходу генератора випадкових підстановок практично без перевірки.

Додатним корисним результатом можна вважати можливість підвищення швидкодії шифрів за рахунок скорочення числа циклів зашифрування.

6. ВИСНОВКИ

Можна зробити загальний висновок, що операція введення циклових підключів практично не впливає на показники випадковості шифру ШУП, та й скоріше і на стійкість багатьох інших шифрів. Основний фактор, що впливає на показники випадковості – це конструкція циклової функції. В нашому випадку була розглянута циклова функція шифру ШУП.

Описаний в роботі новий шифр, названий ШУП-ом і його модифікації, представляються перспективними рішеннями по побудуванню сучасних SPN шифрів. Вони володіють найкращими з відомих шифрів динамічними показниками приходу до стану випадкової підстановки (показниками випадковості). Шифр з двошаровою конструкцію першого циклу стає випадковою підстановкою вже після двох циклів шифрування. За іншими показниками стійкості цей шифр успадкував усі високі показники, властиві шифру «Мухомор» [5]. Відзначимо також важливу особливість шифру, яка полягає в тому, что в шифрі практично без зниження динамічних показників його приходу до стану випадкової підстановки можуть застосовуватися випадкові S-блоки.

Література

[1] Dolgov V.I. The new concept of block symmetric ciphers design / V.I. Dolgov, I.V. Lisitska, K.Ye. Lisitskyi // DOI: 10.1615/TelecomRadEng.v. 76.i. 2. pages 157-184.

[2] J. Daemen and V. Rijmen. AES Proposal: Rijndael. 1st AES Conference, California, USA, 1998. http://www.nist.gov/aes.

[3] P. Junod and S. Vaudenay, FOX: a new family of block ciphers. In H. Handschuh and A. Hasan, editors, Selected Areas in Cryptography: 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004. Revised Selected Papers, volume 3357 of Lecture Notes in Computer Science, pages 114-129. Springer-Verlag, 2004.

[4] Стандарт блочного симметричного преобразования ДСТУ 7624:2014.

[5] Горбенко І.Д. Перспективний блоковий симетричний шифр «Мухомор» – основні положення та специфікація / І.Д. Горбенко, М.Ф. Бондаренко, В.І. Долгов, Р.В. Олійников та інші // Прикладная радиоэлектроника. – Харьков: ХТУРЭ. – 2007. – Том. 6, №2. – С. 147-157.

[6] Информационная технология. Криптографическая защита информации. Блочные шифры. ГОСТ Р 34.12 – 2015. – Москва Стандартинформ. 2015. – 21 с. (Национальный стандарт Российской Федерации).

[7] Государственный стандарт республики Беларусь. СТБ 34.101.31-2011. Информационные технологии. Защита информации Криптографические алгоритмы шифрования и контроля целостности. Введен в действие постановлением Госстандарта Республики Беларусь от 31 января 2011 г. № 5. Изд-во Госстандарт, Минск. – 2011. – 35 с.

[8] Горбенко Ю.І. Аналіз стійкості пост-квантових систем / Ю.І. Горбенко, Р.С. Ганзя // Прикладна радіоелектроніка. – Харків: ХНУРЭ 2014. – Том 13. – №3. С. 268-274.

[9] ISCI'2018: Information Security in critical infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Aleksandr A. Kyznetsov. ASC Academic Pablishin. Mindon, Nevada, USA. / K.E. Lisitsky and others // Chapter 4, Block symmertrik cipher with random S-boxes. p. 100-124.

[10] Лисицкий К.Е. Усовершенствованный Rijndael / К.Е. Лисицкий, // Материалы VI Международной научно-технической конференции "Захист інформації і безпека інформаційних систем". – Львів – 1-2 червня 2017 р. С. 85-86.

[11] Долгов В.И. Усовершенствованный блочный симметричный шифр Калина / В.И. Долгов, И.В. Лисицкая, К.Е. Лисицкий // 0485-8972. – Радиотехника Всеукр. межвед. научн.-техн. сб. – 2016. – Вип.186. – С. 119-131.

[12] Lisitskaya Iryna Impruvd Rijndael / Iryna Lisitskaya, Konstantin Lisitskiy, Mariya Rodinko // Science and Education Studies "Stanford University Press"

Volume II – № 1(17), January- June – 2016. p. 608-618. 25

[13] Пат. 111547 Україна, МПК (2016.01) G09C 1/00 H04L 9/06 (2006.01). Спосіб криптографічного перетворення двійкових даних (варіанти) / Горбенко І.Д., Долгов В.І., Лисицька І.В. та інші (Україна); заявник АО ІІТ м. Харків. № a201500942; заявл. 06.02.2015; опубл. 10.05.2016, Бюл. № 9. – 20 с. 23

[14] Пат. 111448 Україна, МПК H04L 29/14 (2006.01) H04L 9/14 (2006.01) H04L 9/06 (2006.01).Спосіб криптографічного перетворення двійкових даних / Горбенко І.Д., Долгов В.І., Лисицька І.В. та інші (Україна); заявник АО ІІТ м. Харків. № a201503976; заявл. 25.04.2015; опубл. 25.04.2016, Бюл. № 8 – 20 с. 24

[15] Лисицкая И.В. Большие шифры − случайные подстановки. Сравнение дифференциальных и линейных свойств шифров, представленных на украинский конкурс и их уменьшенных моделей / И.В. Лисицкая, А.А. Настенко, Е.К. Лисицкий // Автоматизированные системы управления и приборы автоматики. – 2012.– Вып.159. – С. 4-10.

Поступила в редакцию . .2019.

Лисицька Ірина Вікторівна, професор ХНУ имені В.Н. Каразіна. Область наукових інтересів: технології блочного симетричного шифрування.

Васильев Василий Александрович, магістрант ХНУ радіоелектроніки. Область научных інтересів: технології блочного симетричного шифрування

УДК 621. 3.06

Исследование показателей случайности шифра ШУП / И.В. Лисицкая, В.А. Васильев // Прикладная радиоэлектроника: научно-техн. Журнал. – 2018. – Том XX. – № X. – С. XXX-XXX.

Изучается механизм активизации S-блоков шифра Шуп, который рассматривается как перспективное решение по построению современного блочного симметричного шифра. Приводятся результаты экспериментов по определению влияния на показатели стойкости шифра операций ввода цикловых подключей. Делается вывод, что эти операции не изменяют минимальное количество S-блоков, активизируются на первых циклах шифрование, то есть показатели стойкости шифра не изменяются при различных операциях ввода цикловых подключей. Рассматриваются дифференциальные показатели шифров ШУП-1М с различными S-блоками. Обсуждается конструкция цикловой функции шифра ШУП с двумя слоями подстановочных преобразований. Анализируются минимальные количества активизуемих S-блоков на первых циклах, которые могут быть реализованы в конструкциях шифров ШУП и модернизированных шифров, построенных с использованием управляемых подстановок. Можно сделать вывод, что описанные в работе шифры представляются перспективными решениями по построению современных SPN шифров.

Ключевые слова: активные S-блоки, динамические показатели прихода шифра к состоянию случайной подстановки, дополнительное смешивающее линейное преобразование, распределение полных дифференциалов, распределение максимумов смещений ЛАТ, мини-версия шифра Rijndael, шифр с управляемыми подстановками, усовершенствованный первый цикл

Табл. 4. Рис. 2. Библиогр. 15 наим.

UDC 621. 3.06

Reseaerh indices of SHUP cipher randomness / I.V. Lisitskaya V.A. Vasiliev // Applied radio electronics: scientific and technical. Magazine. – 2019. – Volume XX. – No. X. - P. XXX-XXX.

The mechanism of activation of S-blocks of the SHUP cipher is studied, which is considered as a promising solution for the construction of a modern block symmetric cipher. The results of experiments to determine the impact on the stability of the code of operations of the introduction of loop subkeys are given. It is concluded that these operations do not change the minimum number of S-blocks that are activated on the first cycles of encryption, that is, the cipher stability indicators do not change during different operations of input loop subkeys. Differential indicators of ShUP-1M ciphers with different S-blocks are considered. The design of the cycle function of the SHUP cipher with two layers of substitution transformations is discussed. The minimum quantities of activated S-blocks on the first cycles that can be implemented in the construction of the SUP ciphers and upgraded ciphers constructed using controlled substitutions are analyzed. It is concluded that the ciphers described in the paper appear to be promising solutions for the construction of modern SPN ciphers.

Key words: active S-blocks, dynamic indexes of arrival of the cipher to the state of random substitution, additional mixes linear transformation, distribution of complete differentials, distribution of maximum displacements of LAT, mini version of the cipher Rijndael, cipher with controlled substitutions, advanced first cycle.

Tabl. 4. Fig. 2. Ref. 15 names

№ док.	Позначення	Найменування	Дод. відом.
1		Текстові документи	
	ГЮІК.ХХХХХ1649Ст.01ПЗ	Пояснювальна записка	__ с.
2		Графічні документи	
		Презентаційний матеріал	__ А4.
3		Інші документи	
		Електронна версія ПЗ	1 шт.
		Відзив керівника	1 с.
		Рецензія	1 с.

Изм.	Лист	№ докум.	Підпис	Дата	ГЮІК.ХХХХХ1649Ст.01ВД		
Розроб.		Васильєв В.О.			Диференційні властивості блокового симетричного шифру з керованими підстановками	Лит.	Лист 1 / Листів 1
Перевір.		Лисицька І.В.				Т	
Н.контр.		Конєва Н.Ф.				ХНУРЕ Кафедра БІТ	
Затверд.		Халімов Г.З.					