Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчет по лабораторной работе №2
“ Симметричная криптография. Симметричная криптография. СТБ 34.101.31-2011”

Выполнил:
студент гр. 053504
Горожанкин В.О.

Проверил:
ассистент каф. информатики
Лещенко Евгений Александрович

Минск 2023

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

В данной лабораторной работе необходимо реализовать программные средства шифрования и дешифрования текстовых файлов при помощи стандарта шифрования СТБ 34.101.31-2011. В соответствии с вариантом шифрование и дешифрование должно работать в режиме простой замены. Программное средство должно быть реализовано на языке программирования Python.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Входными данными алгоритмов зашифрования и расшифрования являются блок $X \in \{0, 1\}128$ и ключ $\theta \in \{0, 1\}256$.

Выходными данными является блок $Y \in \{0, 1\}128$ — результат зашифрования либо расшифрования слова X на ключе θ: $Y = F\theta(X)$ либо $Y = F -1$ θ (X).

Входные данные подготавливаются следующим образом:

1 Слово X записывается в виде $X = X1 \| X2 \| X3 \| X4$, где $Xi \in \{0, 1\}32$.

2 Ключ θ записывается в виде $\theta = \theta1 \| \theta2 \| \ldots \| \theta8$, $\theta i \in \{0, 1\}32$, и определяются

тактовые ключи $K1 = \theta1$, $K2 = \theta2$, . . . , $K8 = \theta8$, $K9 = \theta1$, $K10 = \theta2$, . . . , $K56 = \theta8$.

Подстановка H. Подстановка $H : \{0, 1\}8 \rightarrow \{0, 1\}8$ задается таблицей 2. В таблице используется шестнадцатеричное представление слов $u \in \{0, 1\}8$. Если u = IJ16, то значение $H(u)$ находится на пересечении строки I и столбца J. Например, H(A216) = 9B16. На рисунке 1 показана таблица подстановки.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	B1	94	BA	C8	0A	08	F5	3B	36	6D	00	8E	58	4A	5D	E4
1	85	04	FA	9D	1B	B6	C7	AC	25	2E	72	C2	02	FD	CE	0D
2	5B	E3	D6	12	17	B9	61	81	FE	67	86	AD	71	6B	89	0B
3	5C	B0	C0	FF	33	C3	56	B8	35	C4	05	AE	D8	E0	7F	99
4	E1	2B	DC	1A	E2	82	57	EC	70	3F	CC	F0	95	EE	8D	F1
5	C1	AB	76	38	9F	E6	78	CA	F7	C6	F8	60	D5	BB	9C	4F
6	F3	3C	65	7B	63	7C	30	6A	DD	4E	A7	79	9E	B2	3D	31
7	3E	98	B5	6E	27	D3	BC	CF	59	1E	18	1F	4C	5A	B7	93
8	E9	DE	E7	2C	8F	0C	0F	A6	2D	DB	49	F4	6F	73	96	47
9	06	07	53	16	ED	24	7A	37	39	CB	A3	83	03	A9	8B	F6
A	92	BD	9B	1C	E5	D1	41	01	54	45	FB	C9	5E	4D	0E	F2
B	68	20	80	AA	22	7D	64	2F	26	87	F9	34	90	40	55	11
C	BE	32	97	13	43	FC	9A	48	A0	2A	88	5F	19	4B	09	A1
D	7E	CD	A4	D0	15	44	AF	8C	A5	84	50	BF	66	D2	E8	8A
E	A2	D7	46	52	42	A8	DF	B3	69	74	C5	51	EB	23	29	21
F	D4	EF	D9	B4	3A	62	28	75	91	14	10	EA	77	6C	DA	1D

Рисунок 1 – Таблица подстановки H

Преобразования Gr (r = 5, 13, 21). Преобразование Gr : {0, 1}32 → {0, 1}32 ставит в соответствие слову $u = u1 \| u2 \| u3 \| u4$, $ui \in$ {0, 1}8, слово $Gr(u)$ = RotHir $(H(u1) \| H(u2) \| H(u3) \| H(u4))$.

Переменные. Используются переменные a, b, c, d, e со значениями из {0, 1}32.

На рисунке 2 показана схема зашифрования.

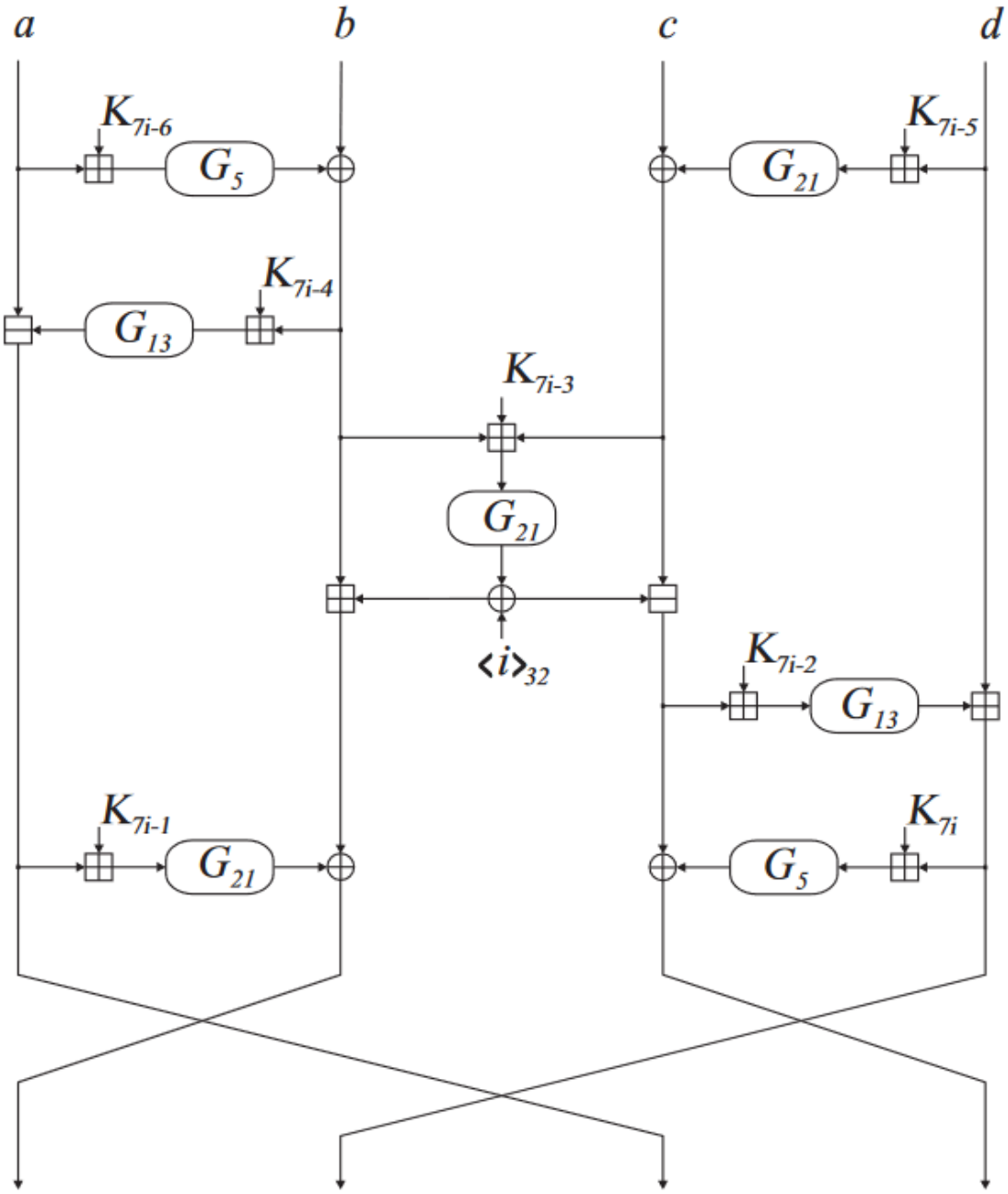

Рисунок 2 – Схема зашифрования

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА

В качестве ключа шифрования используется строка *abcdefghijlmnop*. На вход подается сообщение, кратное 128 битам. В результате работы программы на экран выводится зашифрованное и расшифрованное сообщение. Вывод показан на рисунке 3.

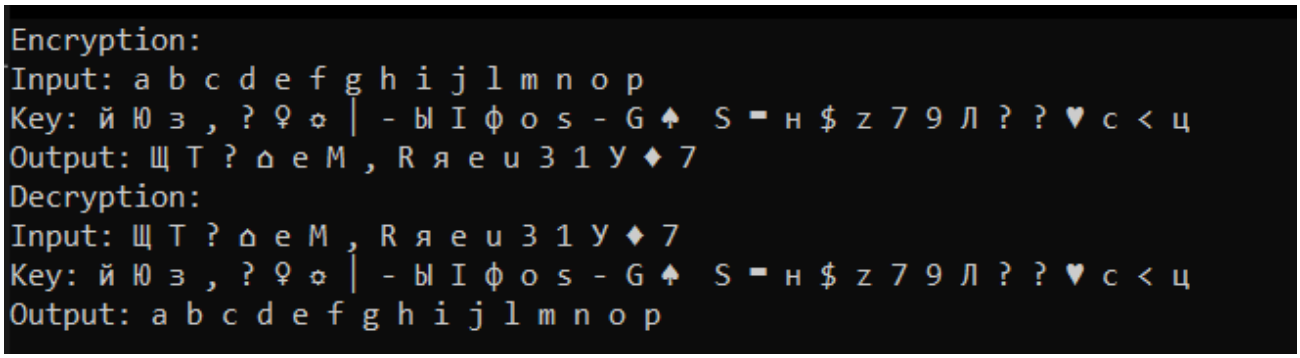

Рисунок 3 – Результат работы программы

4 ВЫВОД

СТБ 34.101.31-2011 определяет семейство криптографических алгоритмов шифрования и контроля целостности, которые используются для защиты информации при ее хранении, передаче и обработке. Настоящий стандарт применяется при разработке средств криптографической защиты информации.

5 ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import struct

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

```
KeyIndex = [
    [0, 1, 2, 3, 4, 5, 6],
    [7, 0, 1, 2, 3, 4, 5],
    [6, 7, 0, 1, 2, 3, 4],
    [5, 6, 7, 0, 1, 2, 3],
    [4, 5, 6, 7, 0, 1, 2],
    [3, 4, 5, 6, 7, 0, 1],
    [2, 3, 4, 5, 6, 7, 0],
    [1, 2, 3, 4, 5, 6, 7]
]


def to_4bytes(value):
    four_byte_value = value & 0xFFFFFFFF  # Применение маски для
ограничения в 4 байта
    # Шаг 2: Применение операции двоичного дополнения (Two's
complement)
    if four_byte_value >= 2 ** 31:
        four_byte_value -= 2 ** 32
    return four_byte_value


def ROTL32(x, r):
    return ((x << r) | (x >> (32 - r)))


def HU1(x, H):
    return (H[((x >> 24) & 0xff)] << 24)


def HU2(x, H):
    return (H[((x >> 16) & 0xff)] << 16)


def HU3(x, H):
    return (H[((x >> 8) & 0xff)] << 8)


def HU4(x, H):
    return (H[((x >> 0) & 0xff)] << 0)
```

```
def G(x, H, r):
    return ROTL32(HU4(x, H) | HU3(x, H) | HU2(x, H) | HU1(x, H), r)


def SWAP(x, y):
    return y, x


def load32(data):
    return struct.unpack("<I", data)[0]


def store32(value):
    p = bytearray(4)  # Создаем байтовый массив длиной 4 байта (32
бита)
    p[0] = (value >> 0) & 0xFF
    p[1] = (value >> 8) & 0xFF
    p[2] = (value >> 16) & 0xFF
    p[3] = (value >> 24) & 0xFF
    return p


def belt_init(ks, k, klen):
    for i in range(32):
        ks[i] = 0

    key = [0] * 8

    for i in range(8):
        key[i] = load32(k[4 * i:4 * (i + 1)])

    if klen == 16:
        for i in range(16):
            ks[i] = k[i]
            ks[i + 16] = k[i]
    elif klen == 24:
        for i in range(24):
            ks[i] = k[i]
        ks[24:28]    =    struct.pack("<I",    load32(k[0:4])    ^
load32(k[4:8]) ^ load32(k[8:12]))
        ks[28:32]    =    struct.pack("<I",    load32(k[12:16])    ^
load32(k[16:20]) ^ load32(k[20:24]))
```

```
    elif klen == 32:
        for i in range(32):
            ks[i] = k[i]


def belt_encrypt(out, _in, ks):
    a = load32(_in[0:4])
    b = load32(_in[4:8])
    c = load32(_in[8:12])
    d = load32(_in[12:16])
    e = 0
    key = [0] * 8

    for i in range(8):
        key[i] = load32(ks[4 * i:4 * (i + 1)])

    for i in range(8):
        b = to_4bytes(b ^ G((a + key[KeyIndex[i][0]]), H, 5))
        c = to_4bytes(c ^ G((d + key[KeyIndex[i][1]]), H, 21))
        a = to_4bytes(a - G((b + key[KeyIndex[i][2]]), H, 13))
        e = to_4bytes(G((b + c + key[KeyIndex[i][3]]), H, 21) ^ (i +
1))
        b = to_4bytes(b + e)
        c = to_4bytes(c - e)
        d = to_4bytes(d + G((c + key[KeyIndex[i][4]]), H, 13))
        b = to_4bytes(b ^ G((a + key[KeyIndex[i][5]]), H, 21))
        c = to_4bytes(c ^ G((d + key[KeyIndex[i][6]]), H, 5))
        a, b = SWAP(a, b)
        c, d = SWAP(c, d)
        b, c = SWAP(b, c)

    out[0:4] = store32(b)
    out[4:8] = store32(d)
    out[8:12] = store32(a)
    out[12:16] = store32(c)


def belt_decrypt(out, _in, ks):
    a = load32(_in[0:4])
    b = load32(_in[4:8])
    c = load32(_in[8:12])
    d = load32(_in[12:16])
```

```
    e = 0
    key = [0] * 8

    for i in range(8):
        key[i] = load32(ks[4 * i:4 * (i + 1)])

    for i in range(8):
        b = to_4bytes(b ^ G((a + key[KeyIndex[7 - i][6]]), H, 5))
        c = to_4bytes(c ^ G((d + key[KeyIndex[7 - i][5]]), H, 21))
        a = to_4bytes(a - G((b + key[KeyIndex[7 - i][4]]), H, 13))
        e = to_4bytes((G((b + c + key[KeyIndex[7 - i][3]]), H, 21) ^
(7 - i + 1)))
        b = to_4bytes(b + e)
        c = to_4bytes(c - e)
        d = to_4bytes(d + G((c + key[KeyIndex[7 - i][2]]), H, 13))
        b = to_4bytes(b ^ G((a + key[KeyIndex[7 - i][1]]), H, 21))
        c = to_4bytes(c ^ G((d + key[KeyIndex[7 - i][0]]), H, 5))
        a, b = SWAP(a, b)
        c, d = SWAP(c, d)
        a, d = SWAP(a, d)

    out[0:4] = store32(c)
    out[4:8] = store32(a)
    out[8:12] = store32(d)
    out[12:16] = store32(b)
```