Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Методы защиты информации

ОТЧЕТ
по лабораторной работе №2
на тему

Симметричная криптография. СТБ 34.101.31-2011

Студент Д. А. Демидова

Проверил Е. А. Лещенко

Минск 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

В данной лабораторной работе необходимо изучить теоретические сведения, а также реализовать программное средство шифрования и дешифрования текстовых файлов при помощи алгоритма СТБ 34.101.31-2011 в режиме сцепления блоков.

2 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Содержимое файла input.txt:
hello my name is dasha
i'm studying in bsuir
lalalalalalalalalalala

Содержимое файла encrypted.txt:
(¥§émS:Z§J \ ÒÂù®_¼3[QbP3 z#è# ´Ð#V)!ê#ÖmØÓ#Êº Z
A##:¼ò¨û 7)# §ó
È¬Ò#òÎ# ï¦ >#

Содержимое файла decrypted.txt:
hello my name is dasha
i'm studying in bsuir
lalalalalalalalalalala

ВЫВОД

В ходе выполнения данной лабораторной работы были изучены теоретические сведения об алгоритме шифрования СТБ 34.101.31-2011 в режимах простой замены, сцепления блоков, гаммирования с обратной связью, счетчика.

Также было реализовано консольное приложение, осуществляющее шифрование и дешифрование содержимого текстовых файлов при помощи стандарта шифрования СТБ 34.101.31-2011 в режиме шифрования методом сцепления блоков.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программного кода

Файл main.py

```
from random import choice
from string import ascii_uppercase

from gost3410131 import STB
from text_helpers import str2bin, bin2str


def main():
    stb = STB()
    with open('text.txt', 'r') as f:
        text = f.read()

    KEY = ''.join(choice(ascii_uppercase) for _ in range(32))

    enc = stb.encrypt(str2bin(text), str2bin(KEY))
    enc_str = bin2str(enc)
    with open('encrypted-text.txt', 'w') as f:
        f.write(enc_str)
    print(f'Файл зашифрован')

    dec = stb.decrypt(enc, str2bin(KEY))
    dec_str = bin2str(dec)
    with open('decrypted-encrypted-text.txt', 'w') as f:
        f.write(dec_str)
    print(f'Файл расшифрован')


if __name__ == '__main__':
    main()
```

Файл gost3410131.py

```
# таблица подстановок
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]
]


class STB:
    def __init__(self):
        self.H = h

    def encrypt(self, data, key):
        tack_keys = extension_key(key)
        all_tack_keys = create_K(tack_keys)
        data = [1] + data
        m = ((len(data) // 128) + 1) * 128
        data = add_padding(data, m)
        res = []
        for i in range(0, m, 128):
            res += self.encrypt_block(data[i:i + 128], all_tack_keys)
        return res

    def decrypt(self, data, key):
        tack_keys = extension_key(key)
        all_tack_keys = create_K(tack_keys)
        res = []
        for i in range(0, len(data), 128):
            res += self.decrypt_128(data[i:i + 128], all_tack_keys)
        while res[0] != 1:
            res = res[1:]
        return res[1:]

    def sum_mod(self, first, second):
```

```
        return (self.to_int(first) + self.to_int(second)) % (2 ** 32)

    def sub_mod(self, first, second):
        sub = self.to_int(first) - self.to_int(second)
        if sub < 0:
            sub += 2 ** 32
        return sub

    def func(self, a, b, c):
        for j in range(32):
            a[j] = b[j] ^ c[j]
        return a

    def to_int(self, lst):
        return int("".join(str(_) for _ in lst), 2)

    def to_list(self, n):
        return [int(i) for i in "{0:b}".format(n)]

    def encrypt_block(self, data, key):
        a = [int(i) for i in data[:32]]
        b = [int(i) for i in data[32:64]]
        c = [int(i) for i in data[64:96]]
        d = [int(i) for i in data[96:]]
        for i in range(8):
            ak = self.g(add_padding(self.to_list(self.sum_mod(a, key[7 * (i)
- 6])), 32), 5)
            b = self.func(b, b, ak)
            # 2
            dk = self.g(add_padding(self.to_list(self.sum_mod(d, key[7 * (i)
- 5])), 32), 21)
            c = self.func(c, c, dk)
            # 3
            bk = self.g(add_padding(self.to_list(self.sum_mod(b, key[7 * (i)
- 4])), 32), 13)
            diff = self.sub_mod(a, bk)
            a = add_padding(self.to_list(diff), 32)
            # 4
            sum_bck = (self.to_int(b) + self.to_int(c) + self.to_int(key[7 *
(i) - 3])) % (2 ** 32)
            bck = self.g(add_padding(self.to_list(sum_bck), 32), 21)
            e = self.func(add_padding([0], 32), bck,
add_padding(self.to_list(i + 1), 32))
            # 5
            b = add_padding(self.to_list(self.sum_mod(b, e)), 32)
            # 6
            c = add_padding(self.to_list(self.sub_mod(c, e)), 32)
            # 7
            ck = self.g(add_padding(self.to_list(self.sum_mod(c, key[7 * (i)
- 2])), 32), 13)
            d = add_padding(self.to_list(self.sum_mod(d, ck)), 32)
            # 8
```

```
            ak = self.g(add_padding(self.to_list(self.sum_mod(a, key[7 * (i)
- 1])), 32), 21)
            b = self.func(b, b, ak)
            # 9
            ck1 = self.g(add_padding(self.to_list(self.sum_mod(d, key[7 *
(i)])), 32), 5)
            c = self.func(c, c, ck1)
            a, b = b, a
            c, d = d, c
            b, c = c, b
        y = b + d + a + c
        return y

    def decrypt_128(self, data, key):
        a = [int(i) for i in data[:32]]
        b = [int(i) for i in data[32:64]]
        c = [int(i) for i in data[64:96]]
        d = [int(i) for i in data[96:]]
        for i in reversed(range(8)):
            # 1
            g_ak = self.g(add_padding(self.to_list(self.sum_mod(a, key[7 *
(i)])), 32), 5)
            b = self.func(b, b, g_ak)
            # 2
            g_dk = self.g(add_padding(self.to_list(self.sum_mod(d, key[7 *
(i) - 1])), 32), 21)
            c = self.func(c, c, g_dk)
            # 3)
            g_bk = self.g(add_padding(self.to_list(self.sum_mod(b, key[7 *
(i) - 2])), 32), 13)
            a = add_padding(self.to_list(self.sub_mod(a, g_bk)), 32)
            # 4
            sum_bck = (self.to_int(b) + self.to_int(c) + self.to_int(key[7 *
(i) - 3])) % (2 ** 32)
            g_bck = self.g(add_padding(self.to_list(sum_bck), 32), 21)
            e = self.func(add_padding([0], 32), g_bck,
add_padding(self.to_list(i + 1), 32))
            # 5
            b = add_padding(self.to_list(self.sum_mod(b, e)), 32)
            # 6
            c = add_padding(self.to_list(self.sub_mod(c, e)), 32)
            # 7
            ck = self.g(add_padding(self.to_list(self.sum_mod(c, key[7 * (i)
- 4])), 32), 13)
            d = add_padding(self.to_list(self.sum_mod(d, ck)), 32)
            # 8
            g_ak = self.g(add_padding(self.to_list(self.sum_mod(a, key[7 *
(i) - 5])), 32), 21)
            b = self.func(b, b, g_ak)
            # 9
            g_dk = self.g(add_padding(self.to_list(self.sum_mod(d, key[7 *
(i) - 6])), 32), 5)
            c = self.func(c, c, g_dk)
            a, b = b, a
```

```
            c, d = d, c
            a, d = d, a
        y = c + a + d + b
        return y

    def g(self, u, r):
        x = [u[i:i + 8] for i in range(0, 32, 8)]
        res = []
        for u_i in x:
            u_right = self.to_int(u_i[:4])
            u_left = self.to_int(u_i[4:])
            num = self.to_list(self.H[u_right][u_left])
            res += add_padding(num, 8)
        func_g = res[r:] + res[:r]
        return func_g


def add_padding(data, k):
    if len(data) <= k:
        zeros_size = k - len(data)
        data2 = [0 for i in range(zeros_size)] + data
        return data2


def create_d(key):
    len_key = len(key)
    if len_key <= 128:
        return 4
    if 128 < len_key <= 192:
        return 6
    if 192 < len_key <= 258:
        return 8


def extension_key(key):
    d_counter = create_d(key)
    ext_key = []
    key = add_padding(key, 256)
    for i in range(0, len(key), 32):
        ext_key.append(key[i:i + 32])
    if d_counter == 4:
        ext_key[4] = ext_key[0]
        ext_key[5] = ext_key[1]
        ext_key[6] = ext_key[2]
        ext_key[7] = ext_key[3]
    if d_counter == 6:
        ext_key[6] = f_ext_key_6(ext_key[0], ext_key[1], ext_key[2])
        ext_key[7] = f_ext_key_6(ext_key[3], ext_key[4], ext_key[5])
    return ext_key


def f_ext_key_6(a, b, c):
    res = add_padding([0], 32)
```

```
    for i in range(32):
        res[i] = a[i] ^ b[i] ^ c[i]
    return res


def create_K(list_keys):
    K = []
    for i in range(7):
        for j in list_keys:
            K.append(j)
    return K
```

Файл text_helpers.py

```
def str2bin(st):
    return [int(i) for i in ''.join('{0:08b}'.format(ord(x), 'b') for x in
st)]


def bin2str(input_list):
    res = ''

    if len(input_list) % 8:
        padding = [0 for _ in range(8 - len(input_list) % 8)]
        input_list = padding + input_list
    for i in range(0, len(input_list), 8):
        x = 0
        for j in range(i, i + 8):
            x = x * 2 + input_list[j]
        res += chr(x)
    return res
```