Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 2

Криптографические алгоритмы шифрования и контроля целостности

Выполнил: Нетецкая Ю.В.

Проверил: Олисейчик В.В.

Минск 2021

**Постановка задачи**

Необходимо реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма СТБ 34.101.31-2011 в различных режимах.

**Описание использованных алгоритмов**

СТБ 34.101.31 – блочный шифр с 256-битным ключом и 8 циклами преобразований, который оперирует 128-битными блоками. Криптографические алгоритмы стандарта построены на основе базовых режимов шифрования блоков данных.

Алгоритмы шифрования, описанные в стандарте:

* режим простой замены;
* режим сцепления блоков;
* режим гаммирования с обратной связью;
* режим счётчика;

**Алгоритм шифрования в режиме простой замены.**

Каждая группа включает алгоритм зашифрования и алгоритм расшифрования с использованием секретного ключа. Предполагается, что стороны будут обмениваться сообщениями, используя один ключ, заранее распределен между ними. В режимах простой замены и сцепления блоков шифруются сообщения, которые содержат хотя бы один блок, а в режимах гаммирования с обратной связью и счётчика — сообщения произвольной длины.

Входные данные и выходные данные:

Входными данными алгоритмов зашифрования и расшифрования являются блок  и ключ 

Выходными данными является блок — результат зашифрования либо расшифрования слова на ключе  либо 

Входные данные для шифрования подготавливаются следующим образом:

* Слово  записывается в виде 
* Ключ записывается в виде  и определяются тактовые ключи

Обозначения и вспомогательные преобразования:

Преобразование ставит в соответствие слову , слово 

– это циклический сдвиг влево на r бит.

 – это операция замены 8-битной входной строки подстановкой.

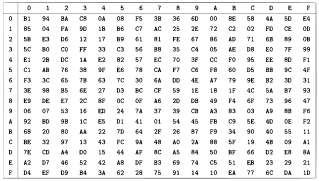


Рисунок 1 – Преобразование Н

Подстановка задается фиксированной таблицей.

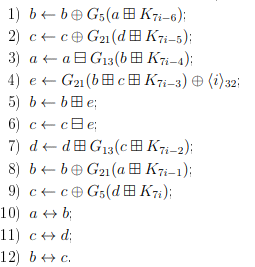
В таблице используется шестнадцатеричное представление слов ,

 и  операции сложения и вычитания по модулю 232.

**Зашифровка:**

Для зашифрования блока  на ключе  выполняются следующие шаги:

* Установить 
* Для i = 1, 2, …, 8 выполнить:



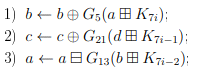
3. Установить 

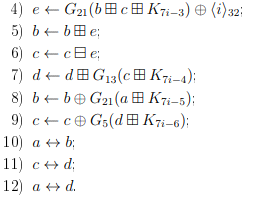
4. Возвратить 

**Расшифровка:**

Для расшифрования блока  на ключе  выполняются следующие шаги:

* Установить 
* Для i = 8, 7, …, 1 выполнить:

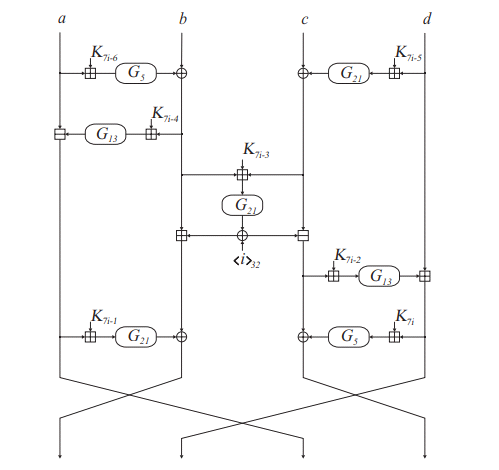




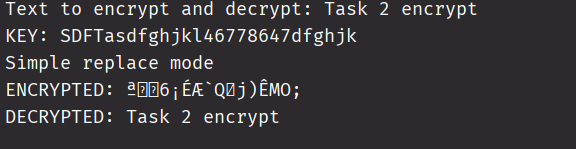
3. Установить  

4. Возвратить  

**3. Блок-схема алгоритма вычислений на i-ом такте шифрования:**



**Результат работы программы**



**Код программы**

def add\_zeros(*data*, *k*):  
 if *len*(*data*) <= *k*:  
 zeros\_size = *k* - *len*(*data*)  
 data2 = [0 for i in *range*(zeros\_size)] + *data* return data2  
  
  
def string\_to\_bin\_list(*st*):  
 bin\_str = ''.join('{0:08b}'.format(*ord*(x), 'b') for x in *st*)  
 ans = [*int*(i) for i in bin\_str]  
 return ans  
  
  
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\_zeros(*key*, 256)  
 for i in *range*(0, *len*(*key*), 32):  
 ext\_key.append(*key*[i:i + 32])  
 if d\_counter == 4:  
 *# продливаем лист* 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:  
 *# добавляем 7 и 8 элементы* 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\_zeros([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  
  
  
class STB:  
 def *\_\_init\_\_*(*self*):  
 *self*.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]  
 ]  
  
 def encrypt\_function(*self*, *data*, *key*):  
 tack\_keys = extension\_key(*key*) *# до 8* all\_tack\_keys = create\_K(tack\_keys) *# до 56* data = [1] + *data* m = ((*len*(*data*) // 128) + 1) \* 128  
 data = add\_zeros(*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\_function(*self*, *data*, *key*):  
 tack\_keys = extension\_key(*key*) *# до 8* all\_tack\_keys = create\_K(tack\_keys) *# до 56* 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\_zeros(*self*.to\_list(*self*.sum\_mod(a, *key*[7 \* (i) - 6])), 32), 5)  
 b = *self*.func(b, b, ak)  
 *# 2* dk = *self*.g(add\_zeros(*self*.to\_list(*self*.sum\_mod(d, *key*[7 \* (i) - 5])), 32), 21)  
 c = *self*.func(c, c, dk)  
 *# 3* bk = *self*.g(add\_zeros(*self*.to\_list(*self*.sum\_mod(b, *key*[7 \* (i) - 4])), 32), 13)  
 diff = *self*.sub\_mod(a, bk)  
 a = add\_zeros(*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\_zeros(*self*.to\_list(sum\_bck), 32), 21)  
 e = *self*.func(add\_zeros([0], 32), bck, add\_zeros(*self*.to\_list(i + 1), 32))  
 *# 5* b = add\_zeros(*self*.to\_list(*self*.sum\_mod(b, e)), 32)  
 *# 6* c = add\_zeros(*self*.to\_list(*self*.sub\_mod(c, e)), 32)  
 *# 7* ck = *self*.g(add\_zeros(*self*.to\_list(*self*.sum\_mod(c, *key*[7 \* (i) - 2])), 32), 13)  
 d = add\_zeros(*self*.to\_list(*self*.sum\_mod(d, ck)), 32)  
 *# 8* ak = *self*.g(add\_zeros(*self*.to\_list(*self*.sum\_mod(a, *key*[7 \* (i) - 1])), 32), 21)  
 b = *self*.func(b, b, ak)  
 *# 9* ck1 = *self*.g(add\_zeros(*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) step 1* g\_ak = *self*.g(add\_zeros(*self*.to\_list(*self*.sum\_mod(a, *key*[7 \* (i)])), 32), 5)  
 b = *self*.func(b, b, g\_ak)  
 *# 2) step 2* g\_dk = *self*.g(add\_zeros(*self*.to\_list(*self*.sum\_mod(d, *key*[7 \* (i) - 1])), 32), 21)  
 c = *self*.func(c, c, g\_dk)  
 *# 3) step 3* g\_bk = *self*.g(add\_zeros(*self*.to\_list(*self*.sum\_mod(b, *key*[7 \* (i) - 2])), 32), 13)  
 a = add\_zeros(*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\_zeros(*self*.to\_list(sum\_bck), 32), 21)  
 e = *self*.func(add\_zeros([0], 32), g\_bck, add\_zeros(*self*.to\_list(i + 1), 32))  
 *# 5* b = add\_zeros(*self*.to\_list(*self*.sum\_mod(b, e)), 32)  
 *# 6* c = add\_zeros(*self*.to\_list(*self*.sub\_mod(c, e)), 32)  
 *# 7* ck = *self*.g(add\_zeros(*self*.to\_list(*self*.sum\_mod(c, *key*[7 \* (i) - 4])), 32), 13)  
 d = add\_zeros(*self*.to\_list(*self*.sum\_mod(d, ck)), 32)  
 *# 8* g\_ak = *self*.g(add\_zeros(*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\_zeros(*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\_zeros(num, 8)  
 func\_g = res[*r*:] + res[:*r*]  
 return func\_g  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 stb = STB()  
 file = *open*("data.txt", "r")  
 text = file.read()  
 file.close()  
 KEY = 'SDFTasdfghjkl46778647dfghjk'  
 *print*('Text to encrypt and decrypt: ' + text)  
 *print*('KEY: ' + KEY)  
 *print*('Simple replace mode')  
 enc = stb.encrypt\_function(string\_to\_bin\_list(text), string\_to\_bin\_list(KEY))  
 *print*('ENCRYPTED: ' + bin\_list\_to\_string(enc))  
 dec = stb.decrypt\_function(enc, string\_to\_bin\_list(KEY))  
 *print*('DECRYPTED: ' + bin\_list\_to\_string(dec))