Министерство образования Республики Беларусь
Белорусский государственный университет информатики и радиоэлектроники
Кафедра информатики

ОТЧЕТ
по лабораторной работе №2

Симметричная криптография. СТБ 34.101.31-2011

Выполнил:
студент гр. 653501
Шинкевич Г. С.

Проверил:
Артемьев В. С.

Минск 2019

ЗАДАНИЕ:

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

Алгоритм СТБ 34.101.31-2011

СТБ 34.101.31 – блочный шифр с 256-битным ключом и 8 циклами криптопреобразований, оперирующий с 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;

Шифрование блока

Входные и выходные данные

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 $\theta : Y = F_\theta(X)$ либо $Y = F_\theta^{-1}(X)$.

Входные данные для шифрования подготавливаются следующим образом:

- Слово X записывается в виде $X = X_1 \| X_2 \| X_3 \| X_4, X_i \in \{0,1\}^{32}$.
- Ключ θ записывается в виде $\theta = \theta_1 \| \theta_2 \| \theta_3 \| \theta_4 \| \theta_5 \| \theta_6 \| \theta_7 \| \theta_8, \theta_i \in \{0,1\}^{32}$ и определяются тактовые ключи $K_1 = \theta_1, K_2 = \theta_2, K_3 = \theta_3, K_4 = \theta_4, K_5 = \theta_5, K_6 = \theta_6, K_7 = \theta_7, K_8 = \theta_8, K_9 = \theta_1, \ldots\ldots, K_{56} = \theta_8.$

Обозначения и вспомогательные преобразования

Преобразование $G_r : \{0,1\}^{32} \to \{0,1\}^{32}$ ставит в соответствие слову $u = u_1 \| u_1 \| u_2 \| u_3 \| u_4, u_i \in \{0,1\}^8$, слово

$G_r(u) = RotHi^r(H(u_1) \parallel H(u_2) \parallel H(u_3) \parallel H(u_4)).$

$RotHi^r$ - циклический сдвиг влево на r бит.

$H(u)$ операция замены 8-битной входной строки подстановкой с рисунка 1.

	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 – Преобразование H

Подстановка $H : \{0,1\}^8 \to \{0,1\}^8$ задается фиксированной таблицей. В таблице используется шестнадцатеричное представление слов $u \in \{0,1\}^8$

$\boxplus$ и $\boxminus$ операции сложения и вычитания по модулю 2^{32}

Зашифрование

Для зашифрования блока X на ключе θ выполняются следующие шаги:

1. Установить $a \leftarrow X_1, b \leftarrow X_2, c \leftarrow X_3, d \leftarrow X_4.$
2. Для i = 1,2,… ,8 выполнить:

1) $b \leftarrow b \oplus G_5(a \boxplus K_{7i-6})$;
2) $c \leftarrow c \oplus G_{21}(d \boxplus K_{7i-5})$;
3) $a \leftarrow a \boxminus G_{13}(b \boxplus K_{7i-4})$;
4) $e \leftarrow G_{21}(b \boxplus c \boxplus K_{7i-3}) \oplus \langle i \rangle_{32}$;
5) $b \leftarrow b \boxplus e$;
6) $c \leftarrow c \boxminus e$;
7) $d \leftarrow d \boxplus G_{13}(c \boxplus K_{7i-2})$;
8) $b \leftarrow b \oplus G_{21}(a \boxplus K_{7i-1})$;
9) $c \leftarrow c \oplus G_5(d \boxplus K_{7i})$;
10) $a \leftrightarrow b$;
11) $c \leftrightarrow d$;
12) $b \leftrightarrow c$.

3. Установить $Y \leftarrow b \| d \| a \| c$.
4. Возвратить Y.

Расшифрование

Для расшифрования блока X на ключе θ выполняются следующие шаги:

1. Установить $a \leftarrow X_1, b \leftarrow X_2, c \leftarrow X_3, d \leftarrow X_4$.
2. Для i = 8,7,… ,1 выполнить:

1) $b \leftarrow b \oplus G_5(a \boxplus K_{7i})$;
2) $c \leftarrow c \oplus G_{21}(d \boxplus K_{7i-1})$;
3) $a \leftarrow a \boxminus G_{13}(b \boxplus K_{7i-2})$;

4) $e \leftarrow G_{21}(b \boxplus c \boxplus K_{7i-3}) \oplus \langle i \rangle_{32}$;
5) $b \leftarrow b \boxplus e$;
6) $c \leftarrow c \boxminus e$;
7) $d \leftarrow d \boxplus G_{13}(c \boxplus K_{7i-4})$;
8) $b \leftarrow b \oplus G_{21}(a \boxplus K_{7i-5})$;
9) $c \leftarrow c \oplus G_5(d \boxplus K_{7i-6})$;
10) $a \leftrightarrow b$;
11) $c \leftrightarrow d$;
12) $a \leftrightarrow d$.

3. Установить $Y \leftarrow c \| a \| d \| b$.
4. Возвратить Y.

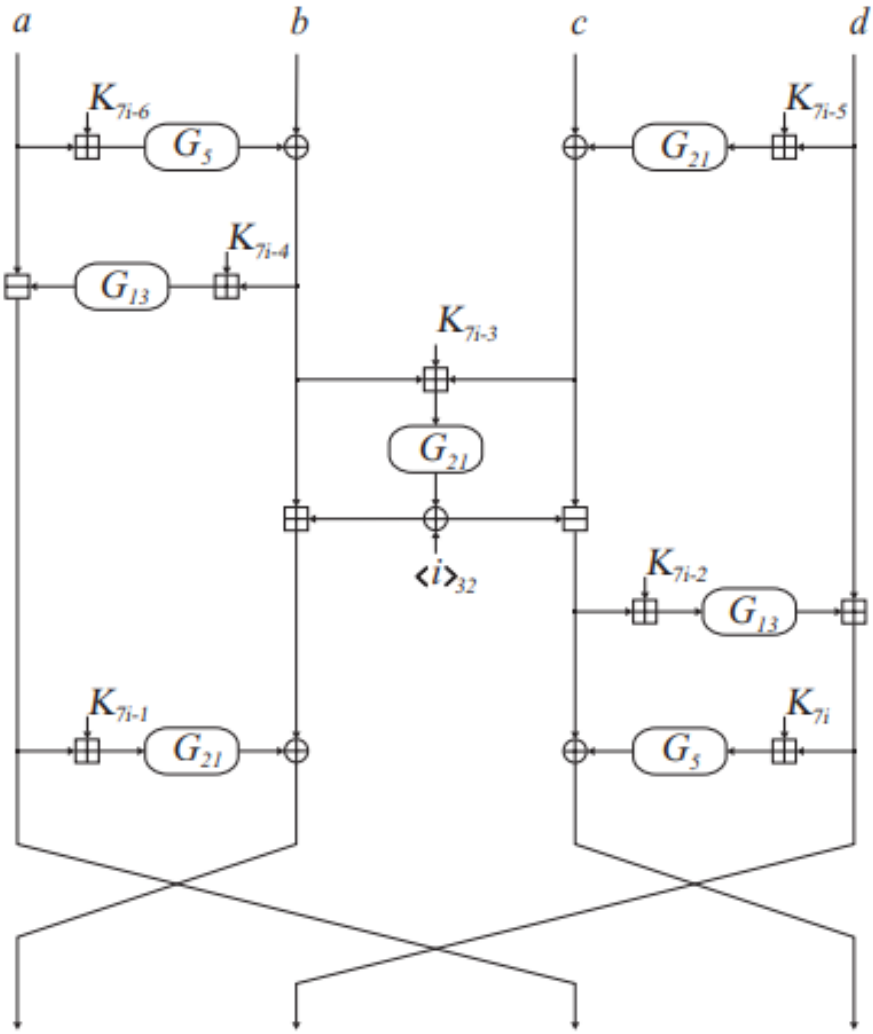

Рисунок 2 – Вычисления на i-м такте шифрования

ДЕМОНСТРАЦИЯ РАБОТЫ

```
Initial text: blablablablabla (626C61626C61626C61626C61626C61)

Encrypted text(ECB): 2C75E9AEFCAEEDDCAAA21EE5816F0B6F
Decrypted text(ECB): blablablablabla (626C61626C61626C61626C61626C61)

Encrypted text(CBC): E42B712F527B81251329B3272B277229
Decrypted text(CBC): blablablablabla (626C61626C61626C61626C61626C61)

Encrypted text(CFB): 1C0C7D98E5123642913A121D78BA6B0A
Decrypted text(CFB): blablablablabla (626C61626C61626C61626C61626C61)

Encrypted text(CTR): 7C170F9D5F7DA3A62D8474F4B3925A7D
Decrypted text(CTR): blablablablabla (626C61626C61626C61626C61626C61)
```

ВЫВОД

В результате лабораторной работы была написана программа для шифрования и дешифрования текстовых файлов с помощью алгоритма СТБ 34.101.31-2011 в режимах простой замены, сцепления блоков, гаммирования с обратной связью и счетчика.

ЛИСТИНГ ПРОГРАММЫ

```
import random
import string

H_TABLE = [
    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 hex_to_bin(hex_string, size):
    return bin(int(hex_string, 16))[2:].zfill(size)
```

```
def bin_to_hex(binary_string):
    return '%0*X' % ((len(binary_string) + 3) // 4, int(binary_string, 2))


def str_to_bin(text):
    return ''.join(format(ord(char), 'b').zfill(8) for char in text)


def str_to_hex(text):
    return bin_to_hex(str_to_bin(text))


def bin_to_str(binary_string):
    text = ''
    for i in range(len(binary_string) // 8):
        bin_number = binary_string[i * 8:(i + 1) * 8]
        number = int(bin_number, 2)
        text += chr(number)
    return text


def RotHi(u, r):
    return u[r:] + u[:r]


def H(u):
    return '{0:b}'.format(H_TABLE[int(u, 2)]).zfill(8)


def G(u, r):
    H_chunks = []
    for i in range(4):
        H_chunks.append(H(u[8 * i:8 * (i + 1)]))
    H_u = ''.join(H_chunks)
    return RotHi(H_u, r)


def U_32(number):
    return '{0:b}'.format(int(number % 2 ** 32)).zfill(32)


def plus_32(u, v):
```

```
    return U_32(int(u, 2) + int(v, 2))


def minus_32(u, v):
    return U_32(int(u, 2) - int(v, 2))


def xor_32(u, v):
    return '{0:b}'.format(int(u, 2) ^ int(v, 2)).zfill(32)


def xor(u, v):
    return '{0:b}'.format(int(u, 2) ^ int(v, 2)).zfill(128)


def U(number):
    return '{0:b}'.format(int(number % 2 ** 128)).zfill(128)


def plus(u, v):
    return U(int(u, 2) + int(v, 2))


def encrypt_block(block, key):
    X = list(chunks(block, 32))
    theta = list(chunks(key, 32))
    K = []
    for i in range(56):
        K.append(theta[i % len(theta)])

    a, b, c, d = X[0], X[1], X[2], X[3]
    for i in range(1, 9):
        b = xor_32(b, G(plus_32(a, K[7 * i - 7]), 5))
        c = xor_32(c, G(plus_32(d, K[7 * i - 6]), 21))
        a = minus_32(a, G(plus_32(b, K[7 * i - 5]), 13))
        e = xor_32(G(plus_32(b, plus_32(c, K[7 * i - 4])), 21), U_32(i))
        b = plus_32(b, e)
        c = minus_32(c, e)
        d = plus_32(d, G(plus_32(c, K[7 * i - 3]), 13))
        b = xor_32(b, G(plus_32(a, K[7 * i - 2]), 21))
        c = xor_32(c, G(plus_32(d, K[7 * i - 1]), 5))
        a, b = b, a
        c, d = d, c
        b, c = c, b
```

```
    return b + d + a + c


def decrypt_block(block, key):
    X = list(chunks(block, 32))
    theta = list(chunks(key, 32))
    K = []
    for i in range(56):
        K.append(theta[i % len(theta)])

    a, b, c, d = X[0], X[1], X[2], X[3]
    for i in range(8, 0, -1):
        b = xor_32(b, G(plus_32(a, K[7 * i - 1]), 5))
        c = xor_32(c, G(plus_32(d, K[7 * i - 2]), 21))
        a = minus_32(a, G(plus_32(b, K[7 * i - 3]), 13))
        e = xor_32(G(plus_32(b, plus_32(c, K[7 * i - 4])), 21), U_32(i))
        b = plus_32(b, e)
        c = minus_32(c, e)
        d = plus_32(d, G(plus_32(c, K[7 * i - 5]), 13))
        b = xor_32(b, G(plus_32(a, K[7 * i - 6]), 21))
        c = xor_32(c, G(plus_32(d, K[7 * i - 7]), 5))
        a, b = b, a
        c, d = d, c
        a, d = d, a

    return c + a + d + b


def chunks(string, size):
    for i in range(0, len(string), size):
        yield string[i:i + size]


def get_padding(text):
    padding_len = 16 - ((len(text) // 8) % 16)
    return padding_len * '{0:b}'.format(padding_len).zfill(8)


def remove_padding(text):
    padding_len = int(text[-8:], 2)
    return text[:-padding_len * 8]
```

```
def ecb_encrypt(text, key):
    text += get_padding(text)
    X = list(chunks(text, 128))

    encrypted_parts = []

    for block in X:
        encrypted_parts.append(encrypt_block(block, key))

    return ''.join(encrypted_parts)


def ecb_decrypt(text, key):
    X = list(chunks(text, 128))

    decrypted_parts = []

    for block in X:
        decrypted_parts.append(decrypt_block(block, key))

    return bin_to_str(remove_padding(''.join(decrypted_parts)))


def cbc_encrypt(text, key, s):
    text += get_padding(text)
    X = list(chunks(text, 128))

    encrypted_parts = []

    encrypted_block = s
    for block in X:
        encrypted_block = encrypt_block(xor(block, encrypted_block), key)
        encrypted_parts.append(encrypted_block)

    return ''.join(encrypted_parts)


def cbc_decrypt(text, key, s):
    X = list(chunks(text, 128))

    decrypted_parts = []

    encrypted_block = s
    for block in X:
```

```
        decrypted_parts.append(xor(encrypted_block, decrypt_block(block, key)))
        encrypted_block = block

    return bin_to_str(remove_padding(''.join(decrypted_parts)))


def cfb_encrypt(text, key, s):
    text += get_padding(text)
    X = list(chunks(text, 128))

    encrypted_parts = []

    encrypted_block = s
    for block in X:
        encrypted_block = xor(encrypt_block(encrypted_block, key), block)
        encrypted_parts.append(encrypted_block)

    return ''.join(encrypted_parts)


def cfb_decrypt(text, key, s):
    X = list(chunks(text, 128))

    decrypted_parts = []

    encrypted_block = s
    for block in X:
        decrypted_parts.append(xor(encrypt_block(encrypted_block, key), block))
        encrypted_block = block

    return bin_to_str(remove_padding(''.join(decrypted_parts)))


def ctr_encrypt(text, key, s):
    text += get_padding(text)
    X = list(chunks(text, 128))

    encrypted_parts = []

    counter = encrypt_block(s, key)
    for block in X:
        counter = plus(counter, U(1))
        encrypted_parts.append(xor(block, encrypt_block(counter, key)))
```

```
    return ''.join(encrypted_parts)


def ctr_decrypt(text, key, s):
    X = list(chunks(text, 128))

    decrypted_parts = []

    counter = encrypt_block(s, key)
    for block in X:
        counter = plus(counter, U(1))
        decrypted_parts.append(xor(block, encrypt_block(counter, key)))

    return bin_to_str(remove_padding(''.join(decrypted_parts)))

def get_init_vector():
    return ''.join(random.choice(string.ascii_letters + string.digits) for x in
range(random.randint(16, 16)))
if __name__ == '__main__':
    with open('text.txt', 'r') as file:
        t = file.read()
    k_x = '12345678987654321234567898765432'
    s = get_init_vector()
    print("Initial text: {} ({})".format(t, str_to_hex(t)))
    print()
    e = ecb_encrypt(str_to_bin(t), str_to_bin(k_x))
    print("Encrypted text(ECB): {}".format(bin_to_hex(e)))
    d = ecb_decrypt(e, str_to_bin(k_x))
    print("Decrypted text(ECB): {} ({})".format(d, str_to_hex(d)))
    print()
    e_cbc = cbc_encrypt(str_to_bin(t), str_to_bin(k_x), str_to_bin(s))
    print("Encrypted text(CBC): {}".format(bin_to_hex(e_cbc)))
    d_cbc = cbc_decrypt(e_cbc, str_to_bin(k_x), str_to_bin(s))
    print("Decrypted text(CBC): {} ({})".format(d_cbc, str_to_hex(d_cbc)))
    print()
    e_cfb = cfb_encrypt(str_to_bin(t), str_to_bin(k_x), str_to_bin(s))
    print("Encrypted text(CFB): {}".format(bin_to_hex(e_cfb)))
    d_cfb = cfb_decrypt(e_cfb, str_to_bin(k_x), str_to_bin(s))
    print("Decrypted text(CFB): {} ({})".format(d_cfb, str_to_hex(d_cfb)))
    print()
    e_ctr = ctr_encrypt(str_to_bin(t), str_to_bin(k_x), str_to_bin(s))
    print("Encrypted text(CTR): {}".format(bin_to_hex(e_ctr)))
    d_ctr = ctr_decrypt(e_ctr, str_to_bin(k_x), str_to_bin(s))
    print("Decrypted text(CTR): {} ({})".format(d_ctr, str_to_hex(d_ctr)))
```