

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ОТЧЕТ
по лабораторной работе
на тему
Симметричная криптография. СТБ 34.101.31-2011

Выполнил студент группы 053501

Криштафович Карина Дмитриевна

Проверил

ассистент кафедры информатики

Лещенко Евгений Александрович

Минск 2023

СОДЕРЖАНИЕ

Введение.....	3
1 Демонстрация работы программы.....	4
1.1 Шифрование.....	4
1.2 Дешифрование.....	4
2 Теоретические сведения.....	5
Заключение.....	8
Приложение А (обязательное) Листинг программного кода.....	9

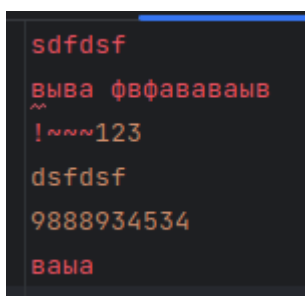
ВВЕДЕНИЕ

Симметричная криптография — это фундаментальная область криптографии, которая занимается шифрованием и дешифрованием данных с использованием одного и того же секретного ключа. В данной лабораторной работе мы будем изучать стандарт криптографии СТБ 34.101.31-2011, который применяется в Республике Беларусь и определяет алгоритмы и принципы симметричного шифрования данных.

В ходе лабораторной работы мы рассмотрим принципы работы алгоритмов шифрования, их использование в защите данных, а также научимся реализовывать шифрование и дешифрование с использованием стандарта СТБ 34.101.31-2011. Эти навыки могут быть полезными в области информационной безопасности и защите конфиденциальных данных.

1 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

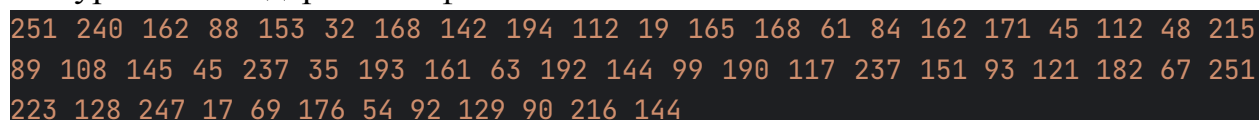
1.1 Шифрование



```
sdfdsf
выва фвфававаыв
^^
!~~~123
dsfdsf
9888934534
вааа
```

Рисунок 1 – Запись исходного текста в файл input.txt

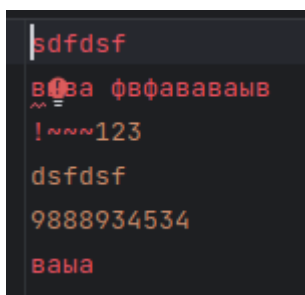
В результате выполнения зашифрованный текст сохраняется в файл encrypted.txt. Содержимое файла:



```
251 240 162 88 153 32 168 142 194 112 19 165 168 61 84 162 171 45 112 48 215
89 108 145 45 237 35 193 161 63 192 144 99 190 117 237 151 93 121 182 67 251
223 128 247 17 69 176 54 92 129 90 216 144
```

1.2 Дешифрование

В результате выполнения расшифрованный текст сохраняется в файл decrypted.txt, что показано на рисунке 2.



```
sdfdsf
выва фвфававаыв
^^
!~~~123
dsfdsf
9888934534
вааа
```

Рисунок 2 – Запись расшифрованного текста в файл decrypted.txt

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Блок-схема алгоритма на i -ом такте шифрования представлена на рисунке 3.

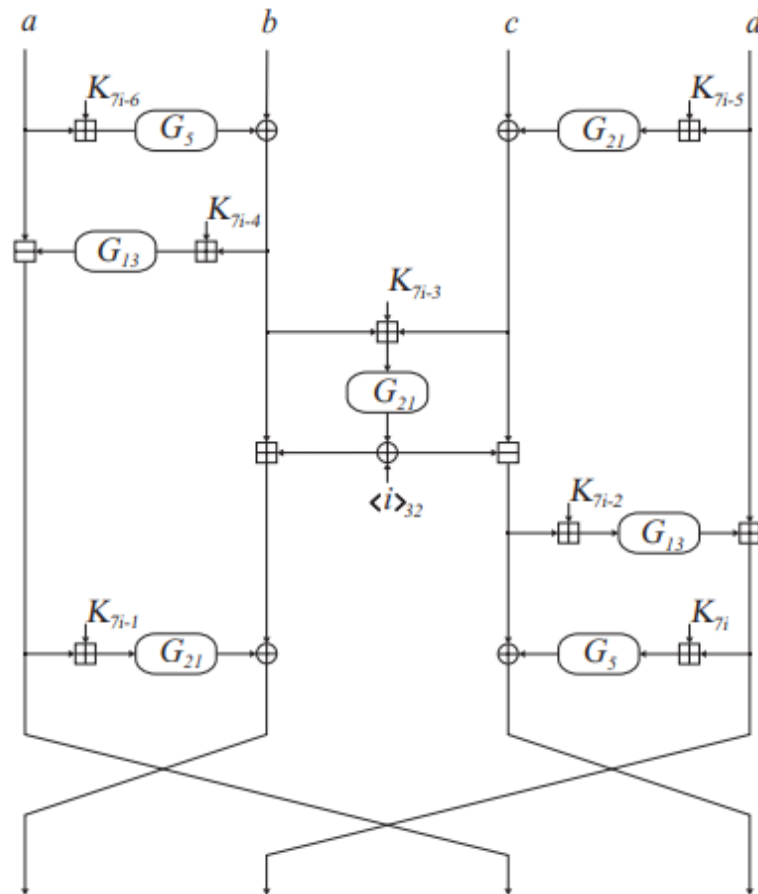


Рисунок 3 – Вычисления на i -м такте шифрования

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 $\theta = \theta_1 \parallel \theta_2 \parallel \theta_3 \parallel \theta_4 \parallel \theta_5 \parallel \theta_6 \parallel \theta_7 \parallel \theta_8, \theta_i \in \{0, 1\}^{32}$ и определяются тактовые ключи $K_1 = \theta_1, K_2 = \theta_2, K_3 = \theta_3, K_4 = \theta_4, K_5 = \theta_5, K_6 = \theta_6, K_7 = \theta_7, K_8 = \theta_8, K_9 = \theta_1, \dots, K_{56} = \theta_8$.

Обозначения и вспомогательные преобразования

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

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

циклический сдвиг влево на r бит.

$H(u)$ операция замены 8-битной входной строки подстановкой с рисунка 4.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	B1	94	8A	08	0A	0B	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	8D	C0	FF	33	C3	56	BB	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	3D	6A	DD	4E	A7	79	9E	B2	3D	31
7	3E	98	85	6E	27	D3	BC	CF	59	1E	18	1F	4C	5A	87	93
8	E9	DE	E7	2C	8F	0C	0F	A6	2D	DB	49	F4	6F	75	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	4B	AD	2A	8B	5F	19	4B	C9	A1
D	7E	CD	A4	D0	15	44	AF	8C	A5	84	50	BF	66	D2	EB	8A
E	A2	D7	46	52	42	AB	DF	83	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 4 – Преобразование H

Подстановка $H : \{0, 1\}^8 \rightarrow \{0, 1\}^8$ 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}

Для зашифрования блока X на ключе θ выполняются следующие шаги:

- Установить $a \leftarrow X_1, b \leftarrow X_2, c \leftarrow X_3, d \leftarrow X_4$.
- Для $i = 1, 2, \dots, 8$ 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 $Y.$

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 $i = 8, 7, \dots, 1$ 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 $Y.$

ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы мы рассмотрели основные аспекты симметричной криптографии согласно стандарту СТБ 34.101.31-2011. Мы изучили основные принципы симметричных алгоритмов шифрования, их ключевые характеристики, а также роль симметрии в процессе обеспечения информационной безопасности.

Был проведен обзор стандарта СТБ 34.101.31-2011, включая его основные положения и требования. Мы изучили поддерживаемые алгоритмы шифрования и хэширования, а также требования к ключам и их генерации в соответствии с данным стандартом.

Понимание и применение стандарта СТБ 34.101.31-2011 в практике имеет важное значение для обеспечения безопасности информационных систем и данных. Этот стандарт предоставляет надежные инструменты для защиты конфиденциальной информации и обеспечения ее целостности.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программного кода

```
import binascii
import random

class STB:
    def __init__(self, key):
        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]

```

```

    key = [self.list_to_int(key[i:i+4]) for i in range(0, len(key), 4)]

```

```

    self.k = [key[i % 8] for i in range(56)]

```

```

# RotHi operation

```

```

def circularleftshift(self, value, k):

```

```

    bitlength = 32

```

```

    return (value << (k % bitlength) & 2 ** bitlength - 1) ^ (value >> (bitlength - k) % bitlength)

```

```

# Represent 32-bit number as list of bytes

```

```

def int_to_list(self, x):

```

```

    return [x >> i & 0xff for i in [24, 16, 8, 0]]

```

```

# Represent list of bytes as 32-bit number

```

```

def list_to_int(self, x):

```

```

    l = [24, 16, 8, 0]

```

```

    return sum([x[i] << l[i] for i in range(4)])

```

```

def modsub(self, x, y):

```

```

    mod = 2**32

```

```

    return (x - y) % mod

```

```

def modadd(self, *x):

```

```

    mod = 2**32

```

```

    res = 0

```

```

    for el in x:

```

```

        res = (res + self.reverse(el)) % mod

```

```

    return self.reverse(res)

```

```

# H-transformation: replacing byte with another value from table
def htransformation(self, x):
    return self.H[x]

# G-transformation
def gtransformation(self, x, k):
    res = self.list_to_int([self.htransformation(i) for i in self.int_to_list(x)])
    return self.reverse(self.circularleftshift(self.reverse(res), k))

def reverse(self, x):
    l = self.int_to_list(x)
    l.reverse()
    return self.list_to_int(l)

def encryption(self, m):
    a, b, c, d = [self.list_to_int(m[i:i+4]) for i in range(0, len(m), 4)]
    for i in range(8):
        b = b ^ self.gtransformation(self.modadd(a, self.k[7*i+0]), 5)
        c = c ^ self.gtransformation(self.modadd(d, self.k[7*i+1]), 21)
        a = self.reverse(self.modsub(self.reverse(a),
                                     self.reverse(self.gtransformation(self.modadd(b, self.k[7*i+2]), 13))))
        e = (self.gtransformation(self.modadd(b, c, self.k[7*i+3]), 21)) ^ self.reverse(i+1)
        b = self.modadd(b, e)
        c = self.reverse(self.modsub(self.reverse(c), self.reverse(e)))
        d = self.modadd(d, self.gtransformation(self.modadd(c, self.k[7*i+4]), 13))
        b = b ^ self.gtransformation(self.modadd(a, self.k[7*i+5]), 21)
        c = c ^ self.gtransformation(self.modadd(d, self.k[7*i+6]), 5)
        a, b = b, a
        c, d = d, c

```

```

        b, c = c, b

    a = self.int_to_list(a)
    b = self.int_to_list(b)
    c = self.int_to_list(c)
    d = self.int_to_list(d)

    return b + d + a + c

def decryption(self, m):
    a, b, c, d = [self.list_to_int(m[i:i+4]) for i in range(0, len(m), 4)]
    for i in reversed(range(8)):
        b = b ^ self.gtransformation(self.modadd(a, self.k[7*i+6]), 5)
        c = c ^ self.gtransformation(self.modadd(d, self.k[7*i+5]), 21)
        a = self.reverse(self.modsub(self.reverse(a),
                                     self.reverse(self.gtransformation(self.modadd(b, self.k[7*i+4]), 13))))
        e = (self.gtransformation(self.modadd(b, c, self.k[7*i+3]), 21) ^ self.reverse(i+1)
        b = self.modadd(b, e)
        c = self.reverse(self.modsub(self.reverse(c), self.reverse(e)))
        d = self.modadd(d, self.gtransformation(self.modadd(c, self.k[7*i+2]), 13))
        b = b ^ self.gtransformation(self.modadd(a, self.k[7*i+1]), 21)
        c = c ^ self.gtransformation(self.modadd(d, self.k[7*i+0]), 5)
        a, b = b, a
        c, d = d, c
        a, d = d, a
    a = self.int_to_list(a)
    b = self.int_to_list(b)
    c = self.int_to_list(c)
    d = self.int_to_list(d)

    return c + a + d + b

def CFB_encrypt(self, msg):

```

```

vec_init = 0x1234567891234567

msg[0] = msg[0] ^ vec_init
msg[0] = self.encryption(msg[0])
for i in range(1, len(msg)):
    msg[i] = msg[i] ^ msg[i - 1]
    msg[i] = self.encryption(msg[i])
return msg

def CFB_decrypt(self, msg):
    vec_init = 0x1234567891234567
    temp_one = msg[0]
    msg[0] = self.decryption(msg[0])
    msg[0] = msg[0] ^ vec_init
    for i in range(1, len(msg)):
        temp_two = msg[i]
        msg[i] = self.decryption(msg[i])
        msg[i] = msg[i] ^ temp_one
        temp_one = temp_two
    return msg

def char_to_int(ch):
    if ord(ch) < 140:
        return ord(ch)
    else:
        return ord(ch) - 900

def int_to_char(i):
    if i < 140:
        return chr(i)
    else:

```

```

        return chr(i + 900)

def main():

    with open("input.txt", "r", encoding='utf-8') as file:

        text = file.read()

    count = 0

    while len(text) % 16:

        text += '0'

        count += 1

    encrypted_result = []

    decrypted_result = []

    for i in range(len(text) // 16):

        arr_text = [char_to_int(item) for item in text[16 * i: 16 * (i + 1)]]

        random_bytes = bytes([random.randint(0, 255) for _ in range(32)])

        hex_str = random_bytes.hex()

        key = list(binascii.unhexlify(hex_str))

        my_stb = STB(key)

        encrypted = my_stb.CFB_encrypt(arr_text)

        encrypted_result.extend(encrypted)

        decrypted_result.extend(my_stb.CFB_decrypt(encrypted))

    encrypted_result = encrypted_result[:len(encrypted_result) - count]

    decrypted_result = decrypted_result[:len(decrypted_result) - count]

    with open("encrypted.txt", "w", encoding='utf-8') as f:

        for item in encrypted_result:

            f.write(str(item) + ' ')

    with open("decrypted.txt", "w", encoding='utf-8') as f:

        for item in decrypted_result:

            f.write(int_to_char(item))

```