Цель работы

Разработать программную реализацию ГОСТ 28.147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования».

Задачи работы

- провести анализ нормативного документа ГОСТ 28.147-89;
- на основе проведённого анализа разработать описание алгоритма;
- разработать программную реализацию алгоритма;
- проверить работоспособность алгоритма.

Ход работы

Описанная в стандарте ГОСТ 28.147-89 модель описывает процессы шифрования и дешифрования. Стандарт устанавливает единый алгоритм криптографического преобразования для систем обработки информации в сетях электронных вычислительных машин (ЭВМ), отдельных вы-числительных комплексах и ЭВМ, который определяет правила шифрования данных и выработки имитовставки.

Структурная схема алгоритма криптографического преобразования (криптосхема) содержит: ключевое запоминающее устройство (КЗУ) на 256 бит, состоящее из восьми 32-разрядных накопителей (X0, X1, X2, X3, X4, X5, X6, X7); четыре 32-разрядных накопителя (N1, N2, N3, N4); два 32-разрядных накопителя (N5, N6) с записанными в них постоянными заполнениями C1, C2; два 32-разрядных сумматора по модулю 2^{32} (СМ1, СМ3); 32-разрядный сумматор поразрядного суммирования по модулю 2 (СМ2); 32-разрядный сумматор по модулю (2^{32} - 1) (СМ4); сумматор по модулю 2(СМ5), ограничение на разрядность сумматора СМ5 не накладывается; блок подстановки (К); регистр циклического сдвига на одиннадцать шагов в сторону старшего разряда (R).

Открытые данные, подлежащие зашифрованию, разбивают на блоки по 64 бита в каждом. В результате получают состояние накопителя N1 и состояние накопителя N2. Алгоритм зашифрования 64-разрядного блока открытых данных в режиме простой замены состоит из 32 циклов. В первом цикле начальное заполнение накопителя N1 суммируется по модулю 2^{32} в сумматоре СМ1 с заполнением накопителя X0 при этом заполнение накопителя N1 сохраняется. Результат суммирования преобразуется в блоке подстановки К и полученный

вектор поступает на вход регистра R, где циклически сдвигается на одиннадцать шагов в сторону старших разрядов. Результат сдвига суммируется поразрядно по модулю 2 в сумматоре СМ2 с 32-разрядным заполнением накопителя N2. Полученный в СМ2 результат записывается в N1, при этом старое заполнение N1 переписывается в N2. Первый цикл заканчивается. Последующие циклы осуществляются аналогично.

В 32 цикле результат из сумматора СМ2 вводится в накопитель N2, а в накопителе N1 сохраняется старое заполнение. Полученные после 32-го цикла зашифрования заполнения накопителей N1 и N2 являются блоком зашифрованных данных, соответствующим блоку открытых данных.

Для программной реализации данного алгоритма шифрования и дешифрования использовался язык программирования Python 3.9.

Пример работы разработанного алгоритма:

Для корректной работы алгоритма необходимо задать исходные данные: таблицу перестановки K, 64-битный блок исходного текста, 256-битный ключ.

```
k_box = (
    (0xC, 0x4, 0x6, 0x2, 0xA, 0x5, 0xB, 0x9, 0xE, 0x8, 0xD, 0x7, 0x0, 0x3, 0xF, 0x1),
    (0x6, 0x8, 0x2, 0x3, 0x9, 0xA, 0x5, 0xC, 0x1, 0xE, 0x4, 0x7, 0xB, 0xD, 0x0, 0xF),
    (0xB, 0x3, 0x5, 0x8, 0x2, 0xF, 0xA, 0xD, 0xE, 0x1, 0x7, 0x4, 0xC, 0x9, 0x6, 0x0),
    (0xC, 0x8, 0x2, 0x1, 0xD, 0x4, 0xF, 0x6, 0x7, 0x0, 0xA, 0x5, 0x3, 0xE, 0x9, 0xB),
    (0x7, 0xF, 0x5, 0xA, 0x8, 0x1, 0x6, 0xD, 0x0, 0x9, 0x3, 0xE, 0xB, 0x4, 0x2, 0xC),
    (0x5, 0xD, 0xF, 0x6, 0x9, 0x2, 0xC, 0xA, 0xB, 0x7, 0x8, 0x1, 0x4, 0x3, 0xE, 0x0),
    (0x8, 0xE, 0x2, 0x5, 0x6, 0x9, 0x1, 0xC, 0xF, 0x4, 0xB, 0x0, 0xD, 0xA, 0x3, 0x7),
    (0x1, 0x7, 0xE, 0xD, 0x0, 0x5, 0x8, 0x3, 0x4, 0xF, 0xA, 0x6, 0x9, 0xC, 0xB, 0x2)
)

input_hex = 0x0011AA33445566FF
key_hex = 0x000011112222333344445555666677778888999AAAABBBBCCCCDDDDEEEEFFFF
```

Рисунок 1 – Исходные данные

Происходит разделение 64-бит текста на 32-битные блоки обратного порядка бит.

```
Plain:        0011aa33445566ff
11001100010101011000100000000000
11111111011001101010101000100010
```

Рисунок 2 – Преобразование исходного текста

Происходит преобразование 256-бит ключа в 32-битные блоки обратного порядка бит.

```
Key:          1111222233334444555566667777888899999aaaabbbbccccddddeeeeffff
10001000100010000000000000000000
11001100110011000100010001000100
10101010101010100010001000100010
11101110111011100110011001100110
10011001100110010001000100010001
11011101110111010101010101010101
10111011101110110011001100110011
11111111111111111011101101110111
```

Рисунок 3 – Преобразование ключа

Далее проходят 32 раунда шифрования, содержание операции СМ1, К, R, СМ2.

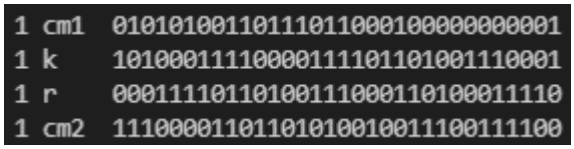

Рисунок 4 – Промежуточные результаты 1 раунда

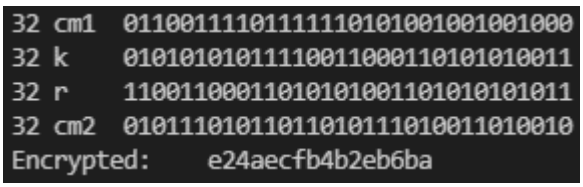

Рисунок 5 – Промежуточные результаты 32 раунда

Процесс расшифрования проходит аналогичным способом, как и шифрование. За исключением порядка чтения ключей из КЗУ.

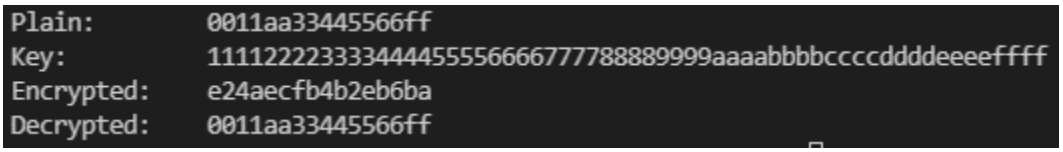

Рисунок 6 – Пример работы программного алгоритма

Выводы

В данной лабораторной работе мы познакомились с алгоритмом цифровой подписи из стандарта ГОСТ 28.147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования», а также проанализировав работу представленного в нормативном документе алгоритма разработали программную реализацию данного алгоритма на языке Python 3.9.

Приложение А. Исходный код разработанного алгоритма.

```
k_box = (
(0xC, 0x4, 0x6, 0x2, 0xA, 0x5, 0xB, 0x9, 0xE, 0x8, 0xD, 0x7, 0x0, 0x3, 0xF, 0x1),
(0x6, 0x8, 0x2, 0x3, 0x9, 0xA, 0x5, 0xC, 0x1, 0xE, 0x4, 0x7, 0xB, 0xD, 0x0, 0xF),
(0xB, 0x3, 0x5, 0x8, 0x2, 0xF, 0xA, 0xD, 0xE, 0x1, 0x7, 0x4, 0xC, 0x9, 0x6, 0x0),
(0xC, 0x8, 0x2, 0x1, 0xD, 0x4, 0xF, 0x6, 0x7, 0x0, 0xA, 0x5, 0x3, 0xE, 0x9, 0xB),
(0x7, 0xF, 0x5, 0xA, 0x8, 0x1, 0x6, 0xD, 0x0, 0x9, 0x3, 0xE, 0xB, 0x4, 0x2, 0xC),
(0x5, 0xD, 0xF, 0x6, 0x9, 0x2, 0xC, 0xA, 0xB, 0x7, 0x8, 0x1, 0x4, 0x3, 0xE, 0x0),
(0x8, 0xE, 0x2, 0x5, 0x6, 0x9, 0x1, 0xC, 0xF, 0x4, 0xB, 0x0, 0xD, 0xA, 0x3, 0x7),
(0x1, 0x7, 0xE, 0xD, 0x0, 0x5, 0x8, 0x3, 0x4, 0xF, 0xA, 0x6, 0x9, 0xC, 0xB, 0x2)
)

input_hex = 0x0011AA33445566FF
key_hex = 0x00001111222233334444555566667777888899999AAAABBBBCCCCDDDDEEEEFFFF

# Разбиение 64-бит блока на 2 блока по 32-бит, записанных в обратном порядке бит
def input_split(input):
    n1 = int(format(input >> 32, '032b')[::-1], 2)
    n2 = int(format(input & 0xFFFFFFFF, '032b')[::-1], 2)
    return n1, n2

# Разбиение 256-бит ключа на 8 блоков 32-бит, записанных в обратном порядке бит
def key_split(key):
    key_list = []
    for i in range(7,-1,-1):
        key_list.append(int(format(((key_hex >> (32 * i)) & 0xFFFFFFFF),
'032b')[::-1], 2))
    return key_list

# Суммирование блока N1 и ключа Xround по модулю 2**32
def cm1_calc(x, n1):
    cm1 = n1 + x
    if cm1 >= 2**32:
        cm1 = cm1 - 2**32 + 1
    cm1 = format(cm1, '032b')
    return cm1

# Замена блоков 4-бит соотстветствущими из таблицы kbox
def kbox_change(cm1):
    k = ''
    for block in range(8):
        k += format(k_box[block][int(cm1[block*4:block*4+4], 2) - 1], '04b')
    return k

# Циклический сдвиг на 11 бит влево
def r_shift(k):
    r = k[11:] + k[:11]
    return r

# Суммирование блока N2 и состояния r поразрядно по модулю 2
def cm2_calc(r, n2):
    cm2 = n2 ^ int(r, 2)
    return cm2
```

```
# Функция шифрования 64-битного блока 256-битным ключом
def encryption(input_hex, key_hex):
    n1, n2 = input_split(input_hex)
    x_list = key_split(key_hex)
    for round in range(1, 25):
        cm1 = cm1_calc(x_list[(round-1) % 8], n1)
        k = kbox_change(cm1)
        r = r_shift(k)
        cm2 = cm2_calc(r, n2)
        n2 = n1
        n1 = cm2
    for round in range(25, 33):
        cm1 = cm1_calc(x_list[(32-round) % 8], n1)
        k = kbox_change(cm1)
        r = r_shift(k)
        cm2 = cm2_calc(r, n2)
        if round == 32:
            n2 = cm2
        else:
            n2 = n1
            n1 = cm2
    cipher = format(n1, '032b')[::-1] + format(n2, '032b')[::-1]
    return cipher

# Функция шифрования 64-битного шифротекста 256-битным ключом
def decryption(cipher, key_hex):
    n1, n2 = input_split(int(cipher, 2))
    x_list = key_split(key_hex)
    for round in range(1, 9):
        cm1 = cm1_calc(x_list[(round-1) % 8], n1)
        k = kbox_change(cm1)
        r = r_shift(k)
        cm2 = cm2_calc(r, n2)
        n2 = n1
        n1 = cm2
    for round in range(9, 33):
        cm1 = cm1_calc(x_list[(32-round) % 8], n1)
        k = kbox_change(cm1)
        r = r_shift(k)
        cm2 = cm2_calc(r, n2)
        if round == 32:
            n2 = cm2
        else:
            n2 = n1
            n1 = cm2
    plain = format(n1, '032b')[::-1] + format(n2, '032b')[::-1]
    return plain

def main():
    # ENCRYPTION
    print('Plain:       ', format(input_hex, '016x'))
    print('Key:         ', format(key_hex, '016x'))
    cipher = encryption(input_hex, key_hex)
    print('Encrypted:   ', format(int(cipher, 2), '016x'))
    # DECRYPTION
    plain = decryption(cipher, key_hex)
    print('Decrypted:   ', format(int(plain, 2), '016x'))

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