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 №1
на тему

Симметричная криптография. Стандарт шифрования ГОСТ 28147-89

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 стандарта шифрования ГОСТ 28147-89 в режиме шифрования методом гаммирования.

2 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

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

Содержимое файла encrypted.txt:
âj895]8Jpñ$e²Á{c ÉkJ#¹lcÒÉ
%J#¹rc ÉaJ#¹rcßÉdJe¹hc ÉhJO¹rcÃÉpJ#¹xcÞÉkJ#¹!
cÞÉkJO¹ccÄÉpJ#¹sc½ÉiJ#¹mcÖÉiJ#¹mcÖÉiJ#¹mcÖÉiJ#¹mcÖÉiJ#¹mcÖÉiJ#

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

ВЫВОД

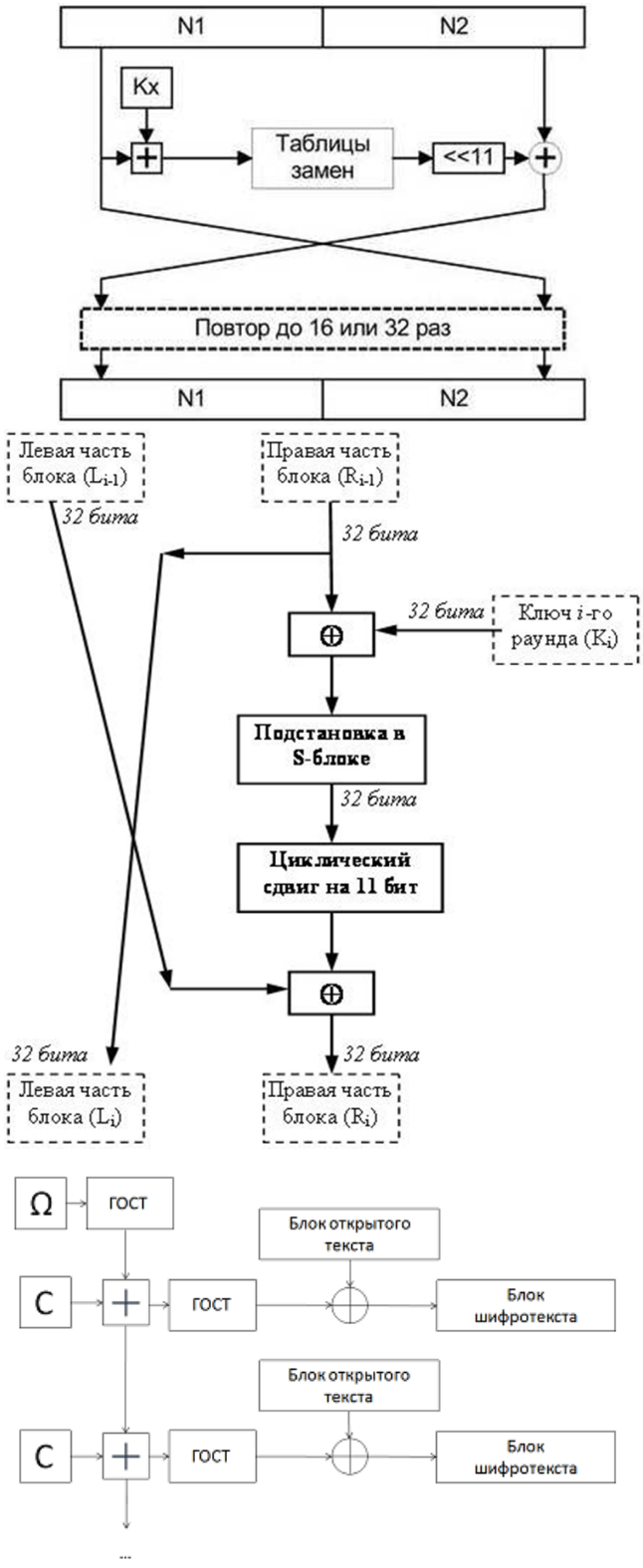
В ходе выполнения данной лабораторной работы были изучены теоретические сведения об алгоритме шифрования ГОСТ 28147-89 в режимах простой замены, гаммирования, гаммирования с обратной связью, генерации имитовставок.

Также было реализовано консольное приложение, осуществляющее шифрование и дешифрование содержимого текстовых файлов с помощью алгоритма при помощи стандарта шифрования ГОСТ 28147-89 в режиме шифрования методом гаммирования.

ПРИЛОЖЕНИЕ А

(обязательное)

Блок-схема алгоритма ГОСТ 28147-89



ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг программного кода

Файл main.py

```
from lab1.gost28147 import encode_file, decode_file


def main():
    encode_file('text.txt')
    decode_file('encrypted-text.txt')


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

Файл gost28147.py

```
from random import choice
from string import ascii_uppercase
from lab1.text_helpers import str_to_binary, text_to_binary, binary_to_text

KEY = ''.join(choice(ascii_uppercase) for _ in range(32))  # ключ (8х32бит)
SYNC = ''.join(choice(ascii_uppercase) for _ in range(8))  # синхропосылка
(начальное состояние гаммы)
# константы для генерации элементов гаммы
C1 = 0x1010101
C2 = 0x1010104
BLOCK_SIZE = 4

# таблица замен
TABLE = [
    (9, 6, 3, 2, 8, 11, 1, 7, 10, 4, 14, 15, 12, 0, 13, 5),
    (3, 7, 14, 9, 8, 10, 15, 0, 5, 2, 6, 12, 11, 4, 13, 1),
    (14, 4, 6, 2, 11, 3, 13, 8, 12, 15, 5, 10, 0, 7, 1, 9),
    (14, 7, 10, 12, 13, 1, 3, 9, 0, 2, 11, 4, 15, 8, 5, 6),
    (11, 5, 1, 9, 8, 13, 15, 0, 14, 4, 2, 3, 12, 7, 10, 6),
    (3, 10, 13, 12, 1, 2, 0, 11, 7, 5, 9, 4, 8, 15, 14, 6),
    (1, 13, 2, 9, 7, 10, 6, 0, 8, 12, 4, 5, 15, 3, 11, 14),
    (11, 10, 15, 5, 0, 12, 14, 8, 6, 2, 3, 9, 1, 7, 13, 4)
]


def prepare_keys():
    # переводим ключ в двоичку
    return str_to_binary(KEY)
```

```
def prepare_sync(text: str):
    # переводим синхропосылку в двоичку
    return str_to_binary(text)


def sum_bits_32(first: bin, second: bin):
    # суммирование по модулю 2^32
    int_sum = int(first, 2) + int(second, 2)

    return format(int_sum, '32b')


def sum_bits_32_1(first: bin, second: bin):
    # суммирование по модулю 2^32 - 1
    int_sum = (int(first, 2) + int(second, 2)) % (2 ** 32 - 1)

    return format(int_sum, '32b')


def get_enc_key(index: int):
    # высчитываем индекс, чтобы брать ключи K1-K8 циклически
    # K1-K24 циклически повторяют K1-K8
    # K25-K32 повторяют K8-K1
    return index % 8 if index < 24 else 7 - index % 8


def move_block_by_table(block: str) -> str:
    # делим на 4-битовые последовательности
    nums = [block[i:i + 4] for i in range(0, len(block), 4)]

    # применяем таблицу замен
    for i in range(len(nums)):
        nums[i] = format(TABLE[i][int(nums[i], 2)], '04b')

    return ''.join(nums)


def xor_values(first: str, second: str) -> str:
    size = len(first)
    result = int(first, 2) ^ int(second, 2)
    result = format(result, f'{size}b')

    return result


def simple_change(text_block: list, keys: list, key_func) -> (str, str):
    left, right = text_block[0], text_block[1]

    # 32 раунда
    for i in range(32):
        # получаем ключи
        key = keys[key_func(i)]
```

```
        # сумма по модулю 2^32 левой части блока с ключом
        enc = sum_bits_32(left, key)[-32:]
        # делим на 4-битовые последовательности, применяем таблицу замен
        enc = move_block_by_table(enc)
        # объединяем выходы таблицы замен в одно 32-битное слово
        enc = int(enc, 2)
        # сдвигаем слово влево на 11 бит
        enc = format((enc << 11), '32b')[-32:]

        # левая часть = ксор левой и правой частей
        enc = xor_values(enc, right)

        # меняем местами части
        if i < 31:
            right = left
            left = enc
        else:
            right = enc

    return left, right


def change_gamma(gamma: list) -> (str, str):
    # суммирование с константами по модулю
    left, right = gamma[0], gamma[1]
    parsed_c1 = format(C1, '32b')
    parsed_c2 = format(C2, '32b')

    new_right = sum_bits_32(right, parsed_c2)[-32:]
    new_left = sum_bits_32_1(left, parsed_c1)[-32:]

    return new_left, new_right


def get_gamma(gamma: list, keys: list) -> (str, str):
    if len(gamma) == 0:
        sync = prepare_sync(SYNC)
        # шифруем синхропосылку алгоритмом простой замены
        # и записываем результат в две части
        left_block, right_block = simple_change(sync, keys, get_enc_key)
    else:
        left_block, right_block = gamma[0], gamma[1]

    # суммируем половинки с константами
    left_block, right_block = change_gamma([left_block, right_block])
    # снова прогоняем половинки через алгоритм простой замены
    left_block, right_block = simple_change([left_block, right_block], keys,
get_enc_key)

    left_block = left_block.replace(' ', '0')
    right_block = right_block.replace(' ', '0')
```

```
    return left_block, right_block


def encode_text(text: str, gamma: list, decode=False) -> (str, str):
    # получаем массив ключей 8x32бита
    keys = prepare_keys()

    new_text = ''.join(text_to_binary(text, decode))
    # получаем гамму
    gamma = get_gamma(gamma, keys)
    gamma = f'{gamma[0]}{gamma[1]}'

    # обработка случаев, когда длина блока не кратна 8/4
    if decode and len(text) % 8 != 0:
        gamma = gamma[:8 * len(text)]
    elif len(text) % 4 != 0:
        gamma = gamma[:16 * len(text)]

    # сумма по модулю 2 с гаммой (тут происходит как шифрование,
    # так и дешифрование, т.к. операция обратна себе)
    encoded_text = xor_values(new_text, gamma)

    return encoded_text.replace(' ', '0'), gamma


def encode_file(filename: str, bit_size=None, decode=False):
    gamma = []

    if decode is True:
        bit_size = 16

    res_file_name = f'lab1/decrypted-{filename}' if decode else
f'lab1/encrypted-{filename}'
    with open(res_file_name, 'w') as output_file:
        with open(f'lab1/{filename}', 'r') as input_file:
            block_size = BLOCK_SIZE
            if decode:
                block_size *= 2

            # обрабатываем файл чанками
            text = input_file.read(block_size)
            while text:
                result, gamma = encode_text(text, gamma, decode)
                res = binary_to_text(result, bit_size)
                output_file.write(res)
                text = input_file.read(block_size)

            input_file.close()

    output_file.close()
```

```
def decode_file(filename: str):
    encode_file(filename, decode=True)
```

Файл text_helpers.py

```
def str_to_binary(s: str) -> list:
    # берем аски код каждого символа в строке и
    # переводим в двоичку (каждый символ длины 8 бит)
    symbols = [
        [format(ord(j), '08b') for j in s[i:i + 4]]
        for i in range(len(s) - 4, -4, -4)
    ]
    bin_repr = [''.join(symbol) for symbol in symbols]

    return bin_repr


def text_to_binary(text: str, decode=False) -> list:
    result = []

    if decode:
        bits = '08b'
    else:
        bits = '16b'

    for txt in text:
        for t in txt:
            result.append(format(ord(t), bits))

    if decode and len(result) > 1:
        new_result = []

        for i in range(0, len(result), 2):
            new_result.append(result[i] + result[i + 1])

        result = new_result

    return [sym.replace(' ', '0') for sym in result]


def binary_to_text(binary: str, bit_size: str) -> str:
    result_str = ''

    if bit_size is None:
        bit_size = 8

    for i in range(0, len(binary), bit_size):
        result_str += chr(int(binary[i:i + bit_size], 2))

    return result_str
```