Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ
к лабораторной работе №1
на тему

СИММЕТРИЧНАЯ КРИПТОГРАФИЯ. СТАНДАРТ ШИФРОВАНИЯ ГОСТ 28147-89

Выполнил: студент гр.253504
Фроленко К.Ю.
Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 ФОРМУЛИРОВКА ЗАДАЧИ

Современное развитие вычислительной техники и повсеместное использование компьютерных сетей привели к тому, что вопросы защиты информации стали одними из наиболее актуальных в области информатики и информационных технологий. Передача и хранение данных требуют не только организационных мер безопасности, но и применения криптографических методов, которые обеспечивают устойчивость информации к несанкционированному доступу.

В данной лабораторной работе ставится задача изучить один из наиболее известных отечественных стандартов симметричного шифрования – ГОСТ 28147-89, и на его основе разработать программное средство, позволяющее выполнять шифрование и дешифрование текстовых файлов. Особенность этой лабораторной работы заключается в том, что необходимо не просто реализовать алгоритм в виде «чёрного ящика», но и организовать возможность наблюдать работу каждого раунда шифрования. Это важно для понимания внутренних принципов построения блочных криптосистем.

Таким образом, цель работы можно сформулировать следующим образом: изучить теоретические основы работы ГОСТ 28147-89, реализовать программный алгоритм в режиме простой замены (ECB), протестировать его на примерах и убедиться в правильности шифрования и расшифрования данных**.**

Для достижения поставленной цели необходимо:

1 Рассмотреть структуру блочного шифра, основанного на схеме Фейстеля.

2 Описать принципы формирования подключей из 256-битового ключа.

3 Изучить особенности S-блоков, обеспечивающих нелинейность преобразования.

4 Реализовать все стадии алгоритма (раундовые преобразования, циклический сдвиг, подстановку, сложение по модулю 2322^{32}232).

5 Организовать процедуру добивки текста, чтобы его длина была кратна размеру блока.

6 Разработать интерфейс командной строки, позволяющий запускать программу в режимах «encrypt» и «decrypt».

7 Провести эксперименты по шифрованию и дешифрованию текстовых файлов и зафиксировать результаты.

В работе рассматривается режим простой замены. Это базовый режим блочных шифров, при котором каждый блок текста шифруется независимо. Его преимущество заключается в простоте реализации, однако у него есть серьёзный недостаток: одинаковые блоки исходного текста преобразуются в одинаковые блоки шифртекста, что позволяет обнаружить статистические закономерности. Несмотря на это, данный режим является удобным объектом для учебных целей и наглядно демонстрирует принципы работы алгоритма.

2 ХОД РАБОТЫ

Для выполнения лабораторной работы был реализован программный комплекс на языке Python, воспроизводящий работу ГОСТ 28147-89 в режиме простой замены. Алгоритм был реализован строго в соответствии с описанием стандарта: блоки текста длиной 64 бита проходили 32 раунда преобразований по схеме Фейстеля, в каждом из которых выполнялись операции сложения по модулю 2^{32}, S-подстановки, циклического сдвига и побитового сложения с левой половиной блока.

В качестве входных данных использовался обычный текстовый файл. Перед шифрованием данные дополнялись байтами по стандарту PKCS#7, что обеспечивало кратность длины блока 8 байтам. Результат шифрования сохранялся в выходной файл в шестнадцатеричном представлении. При расшифровании этот процесс выполнялся в обратном порядке: данные снова делились на блоки, проходили 32 раунда преобразований, но уже с подключами в обратной последовательности, а после этого снималась добивка и текст возвращался в исходный вид.

Отладочный режим программы был реализован таким образом, чтобы пользователь мог наблюдать внутренние состояния алгоритма. После каждого раунда выводились значения полублоков N1 и N2, что позволяло проследить эволюцию данных в процессе шифрования и дешифрования.

```
(venv) PS C:\Workspace\BSUIR-Labs\MZI\Lab1> python main.py decrypt out.txt in.txt

main.py | Блок 00 (DEC) IN: 6524A857F0A3AA46
main.py | Раунд  1: N1=0xF0A3AA46, N2=0x4E17F63B
main.py | Раунд  2: N1=0x4E17F63B, N2=0x7110E1E3
main.py | Раунд  3: N1=0x7110E1E3, N2=0x9CAD5C6E
main.py | Раунд  4: N1=0x9CAD5C6E, N2=0xBDF59861
main.py | Раунд  5: N1=0xBDF59861, N2=0xF7C8377C
main.py | Раунд  6: N1=0xF7C8377C, N2=0xA5F408DF
main.py | Раунд  7: N1=0xA5F408DF, N2=0xEAC4B86E
main.py | Раунд  8: N1=0xEAC4B86E, N2=0x44D57070
main.py | Раунд  9: N1=0x44D57070, N2=0xD58DA501
main.py | Раунд 10: N1=0xD58DA501, N2=0x2A88CC34
main.py | Раунд 11: N1=0x2A88CC34, N2=0xCF06C512
main.py | Раунд 12: N1=0xCF06C512, N2=0xEB9A89D1
main.py | Раунд 13: N1=0xEB9A89D1, N2=0x8D14635E
main.py | Раунд 14: N1=0x8D14635E, N2=0x771D50AC
main.py | Раунд 15: N1=0x771D50AC, N2=0xB134F163
```

Рисунок 1 – Начало работы программы

Примером тестирования стала строка «test». В процессе расшифрования в последних раундах отчётливо отразился блок добивки `0x04040404`, соответствующий корректному PKCS#7. После удаления добивки исходный текст был полностью восстановлен.

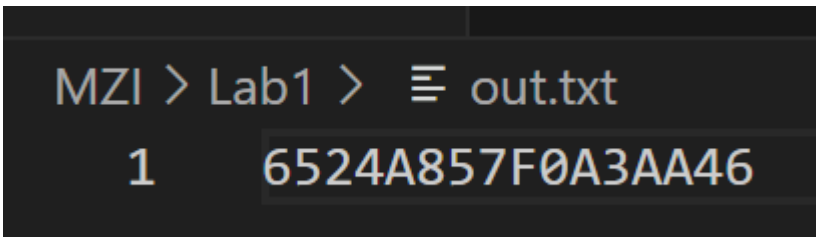

Рисунок 2 – Результат работы программы

Таким образом, было подтверждено, что реализация алгоритма корректна: процедура шифрования и дешифрования полностью обратимы, а все промежуточные данные соответствуют описанию алгоритма ГОСТ 28147-89.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была достигнута основная цель — реализовать на практике один из наиболее известных отечественных стандартов симметричного шифрования ГОСТ 28147-89. В процессе изучения были рассмотрены основные элементы алгоритма: его блочная структура, организация 32 раундов по схеме Фейстеля, порядок формирования подключей из 256-битового ключа, а также роль S-блоков в обеспечении нелинейности преобразований. Особое внимание уделялось пониманию того, как простые арифметические и логические операции в совокупности дают надёжное криптографическое преобразование.

Важным этапом стало то, что программа была реализована не в виде «чёрного ящика», а с возможностью наблюдения всех промежуточных значений. Благодаря этому появилась возможность пошагово проследить работу каждого раунда: от сложения правой половины блока с подключом и подстановки по S-блокам до выполнения циклического сдвига и получения новых значений полублоков. Такой подход позволил глубже понять структуру алгоритма и убедиться, что криптографическая стойкость обеспечивается именно сложной комбинацией простых операций.

Практическая часть продемонстрировала корректность реализации. Исходные текстовые данные были зашифрованы и успешно восстановлены после дешифрования. Появление корректного PKCS#7-паддинга в процессе работы подтвердило правильность организации добивки и обработки блоков фиксированного размера. Проведённые тесты показали полную обратимость алгоритма: результат дешифрования совпал с исходным текстом, что соответствует требованиям симметричных шифров.

Таким образом, поставленные в работе задачи были выполнены полностью. Полученное программное средство можно использовать не только для демонстрации работы ГОСТ 28147-89, но и как учебный инструмент для понимания принципов построения блочных шифров. Лабораторная работа позволила закрепить знания о криптографических методах защиты информации, а также на практике убедиться в том, что теоретические принципы, изложенные в стандарте, корректно реализуются средствами современного программирования.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
import sys
import struct

BLOCK_SIZE = 8
KEY_SIZE = 32

SBOX = [
    [4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3],
    [14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9],
    [5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11],
    [7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3],
    [6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2],
    [4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14],
    [13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12],
    [1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12],
]

KEY = bytes(range(32))


def rotl32(x, n):
    x &= 0xFFFFFFFF
    return ((x << n) & 0xFFFFFFFF) | (x >> (32 - n))


def substitute(x):
    out = 0
    for i in range(8):
        nib = (x >> (4 * i)) & 0xF
        out |= (SBOX[i][nib] & 0xF) << (4 * i)
    return out


def split_block(b):
    return struct.unpack(">II", b)


def join_block(n1, n2):
    return struct.pack(">II", n1 & 0xFFFFFFFF, n2 & 0xFFFFFFFF)


def key_schedule_enc(key):
    k = struct.unpack(">IIIIIIII", key)
    sched = []
    for _ in range(3):
        sched.extend(k)
    sched.extend(reversed(k))
    return sched


def key_schedule_dec(key):
    return list(reversed(key_schedule_enc(key)))


def encrypt_block(block, key, trace=False):
    n1, n2 = split_block(block)
    ks = key_schedule_enc(key)
```

```
    for r in range(32):
        f = (n2 + ks[r]) & 0xFFFFFFFF
        f = substitute(f)
        f = rotl32(f, 11)
        n1, n2 = n2, (n1 ^ f) & 0xFFFFFFFF
        if trace:
            print(f"main.py | Раунд {r+1:2d}: N1=0x{n1:08X}, N2=0x{n2:08X}")
    return join_block(n2, n1)


def decrypt_block(block, key, trace=False):
    n1, n2 = split_block(block)
    ks = key_schedule_dec(key)
    for r in range(32):
        f = (n2 + ks[r]) & 0xFFFFFFFF
        f = substitute(f)
        f = rotl32(f, 11)
        n1, n2 = n2, (n1 ^ f) & 0xFFFFFFFF
        if trace:
            print(f"main.py | Раунд {r+1:2d}: N1=0x{n1:08X}, N2=0x{n2:08X}")
    return join_block(n2, n1)


def pad(data):
    padlen = BLOCK_SIZE - (len(data) % BLOCK_SIZE)
    if padlen == 0:
        padlen = BLOCK_SIZE
    return data + bytes([padlen]) * padlen


def unpad(data):
    if not data or len(data) % BLOCK_SIZE != 0:
        raise ValueError("Некорректная длина для PKCS#7.")
    p = data[-1]
    if p < 1 or p > BLOCK_SIZE or data[-p:] != bytes([p]) * p:
        raise ValueError("Некорректная добивка PKCS#7.")
    return data[:-p]


def safe_ascii(b):
    return "".join(chr(x) if 32 <= x <= 126 else "." for x in b)


def encrypt_text(text):
    data = text.encode("utf-8")
    data = pad(data)
    out = bytearray()
    for i in range(0, len(data), BLOCK_SIZE):
        blk = data[i : i + BLOCK_SIZE]
        print(
            f"\nmain.py    |    Блок    {i//BLOCK_SIZE:02d}    (ENC)    IN:
{blk.hex().upper()}  '{safe_ascii(blk)}'"
        )
        out += encrypt_block(blk, KEY, trace=True)
    return out.hex().upper()


def decrypt_text(hextext):
    hex_clean = "".join(ch for ch in hextext if ch.strip())
    if len(hex_clean) % 2 != 0:
        raise ValueError("Нечетная длина HEX — поврежденный шифртекст?")
    data = bytes.fromhex(hex_clean)
    out = bytearray()
```

```
    for i in range(0, len(data), BLOCK_SIZE):
        blk = data[i : i + BLOCK_SIZE]
        print(f"\nmain.py    |    Блок    {i//BLOCK_SIZE:02d}    (DEC)    IN:
{blk.hex().upper()}")
        out += decrypt_block(blk, KEY, trace=True)
    return unpad(bytes(out)).decode("utf-8")


def usage():
    print("Использование: python main.py encrypt(decrypt) in.txt out.txt")


def main():
    if len(sys.argv) != 4 or sys.argv[1] not in ("encrypt", "decrypt"):
        usage()
        return
    mode, in_path, out_path = sys.argv[1], sys.argv[2], sys.argv[3]

    if mode == "encrypt":
        text = open(in_path, "r", encoding="utf-8").read()
        cipher_hex = encrypt_text(text)
        open(out_path, "w", encoding="utf-8").write(cipher_hex)
        print(f"\nmain.py | Шифртекст (HEX) записан в: {out_path}")
    else:
        hextext = open(in_path, "r", encoding="utf-8").read()
        plain = decrypt_text(hextext)
        open(out_path, "w", encoding="utf-8").write(plain)
        print(f"\nmain.py | Расшифрованный текст записан в: {out_path}")


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