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

БГУИР КП 1-40 04 01 025 ПЗ

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 2^{32}).

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 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» — устаревший государственный стандарт СССР (позже межгосударственный стандарт СНГ), описывающий алгоритм симметричного блочного шифрования и режимы его работы.

Является примером DES-подобных криптосистем, созданных по классической итерационной схеме Фейстеля.

ГОСТ 28147-89 представляет собой симметричный 64-битовый блочный алгоритм с 256-битовым ключом.

Этот алгоритм криптографического преобразования данных предназначен для аппаратной и программной реализации, удовлетворяет криптографическим требованиям и до 1983 года не накладывал ограничений на степень секретности защищаемой информации.

Схема работы алгоритма ГОСТ 28147-89 следующая. Данные, подлежащие зашифровке, разбивают на 64-разрядные блоки.

Эти блоки разбиваются на два субблока N1 и N2 по 32 бит на рисунке 1.

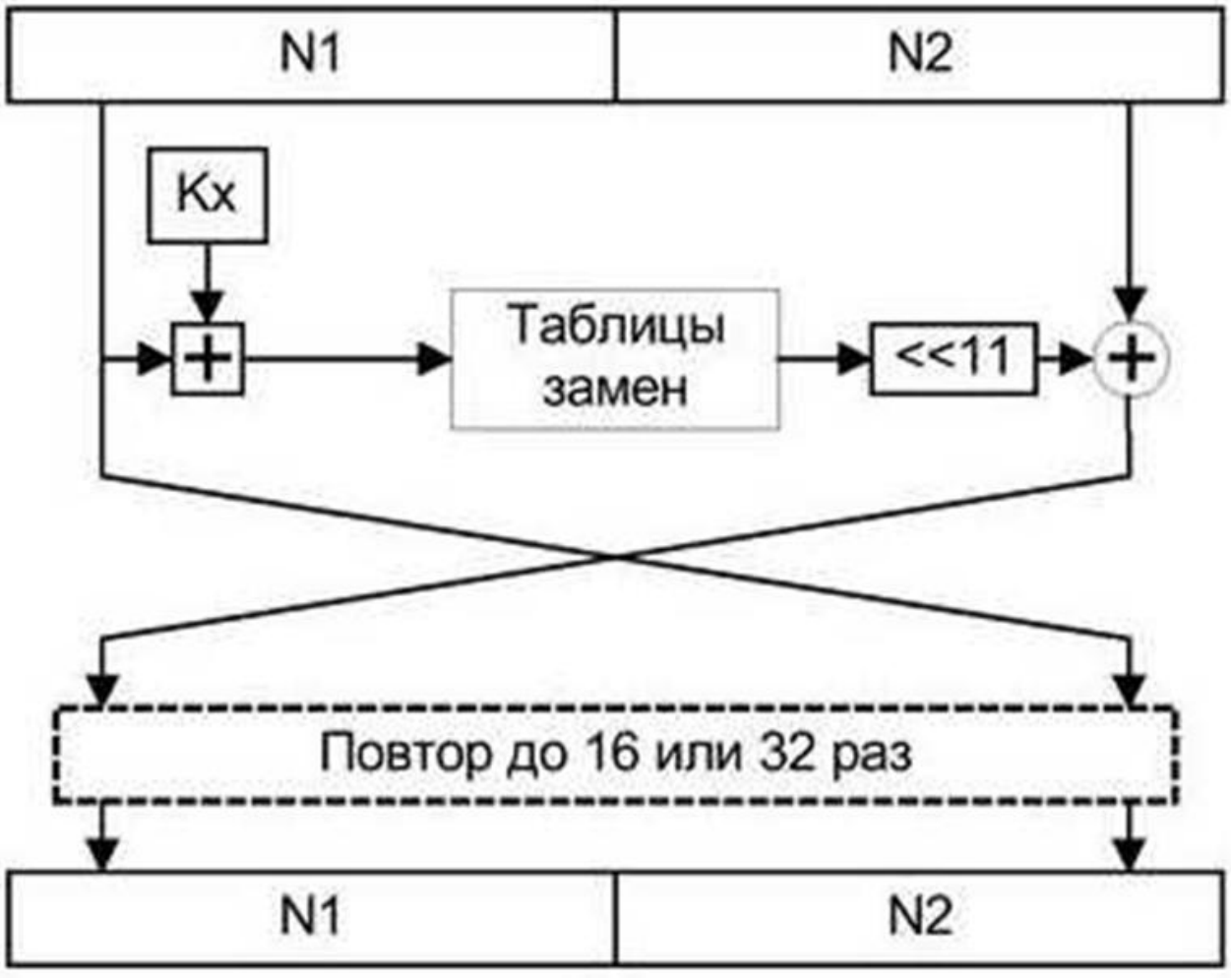

Рисунок 1 – Субблоки N1 и N2

Структура одного раунда ГОСТ 28147-89 на рисунке 2.

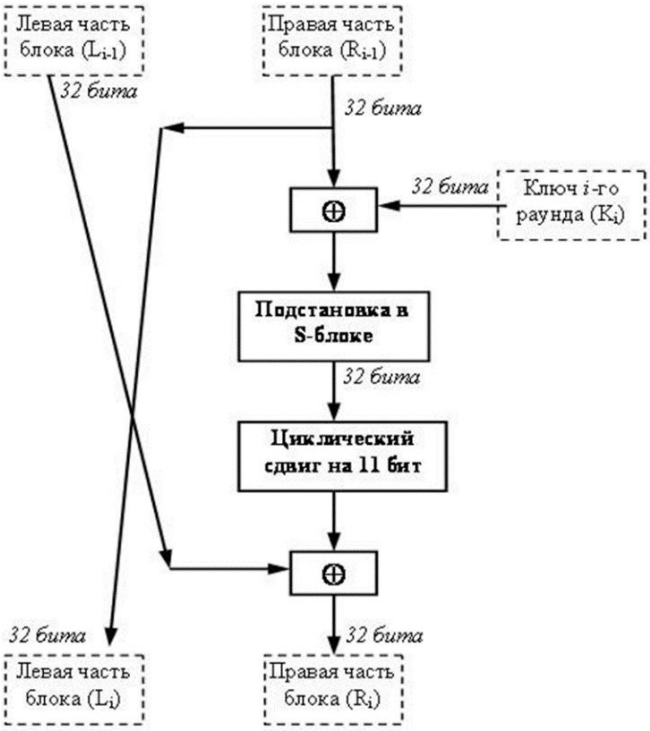

Рисунок 2 – Структура одного раунда

Шифруемый блок данных разбивается на две части, которые затем обрабатываются как отдельные 32-битовые целые числа без знака.

Сначала правая половина блока и подключ раунда складываются по модулю 2^{32}.

Затем производится поблочная подстановка.

32-битовое значение, полученное на предыдущем шаге (обозначим его S), интерпретируется как массив из восьми 4-битовых блоков кода: $S=(S_0,S_1,S_2,S_3,S_4,S_5,S_6,S_7)$.

Далее значение каждого из восьми блоков заменяется на новое, которое выбирается по таблице замен рисунок 3.

Номер S-блока \ x_{ij}	Значение y_{ij}															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
2	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
3	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
4	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
5	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
6	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
7	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
8	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

Рисунок 3 – Таблица замен

В каждой строке таблицы замен записаны числа от 0 до 15 в произвольном порядке без повторений.

Значения элементов таблицы замен взяты от 0 до 15, так как в четырех битах, которые подвергаются подстановке, может быть записано целое число без знака в диапазоне от 0 до 15.

Значение блока S1 (четыре младших бита 32-разрядного числа S) заменится на число, стоящее на позиции, номер которой равен значению заменяемого блока.

Например, в этом случае S1=0 заменится на 4, если S1=1, то оно заменится на 10 и т.д.

После выполнения подстановки все 4-битовые блоки снова объединяются в единое 32-битное слово, которое затем циклически сдвигается на 11 битов влево.

Наконец, с помощью побитовой операции "сумма по модулю 2" результат объединяется с левой половиной, вследствие чего получается новая правая половина Ri.

Новая левая часть Li берется равной младшей части преобразуемого блока: Li= Ri-1.

Полученное значение преобразуемого блока рассматривается как результат выполнения одного раунда алгоритма шифрования.

ГОСТ 28147-89 является блочным шифром, поэтому преобразование данных осуществляется блоками в так называемых базовых циклах.

Базовые циклы заключаются в многократном выполнении для блока данных основного раунда, рассмотренного нами ранее, с использованием разных элементов ключа и отличаются друг от друга порядком использования ключевых элементов.

В каждом раунде используется один из восьми возможных 32-разрядных подключей.

Рассмотрим процесс создания подключей раундов. В ГОСТ эта процедура очень проста, особенно по сравнению с DES. 256-битный ключ K разбивается на восемь 32-битных подключей, обозначаемых K0, K1, K2, K3, K4, K5, K6, K7. Алгоритм включает 32 раунда, поэтому каждый подключ при шифровании используется в четырех раундах в последовательности, представленной в таблице 1.

Таблица 1. Последовательность использования подключей при шифровании

Раунд	1	2	3	4	5	6	7	8
Подключ	K0	K1	K2	K3	K4	K5	K6	K7
Раунд	9	10	11	12	13	14	15	16
Подключ	K0	K1	K2	K3	K4	K5	K6	K7
Раунд	17	18	19	20	21	22	23	24
Подключ	K0	K1	K2	K3	K4	K5	K6	K7
Раунд	25	26	27	28	29	30	31	32
Подключ	K7	K6	K5	K4	K3	K2	K1	K0

Процесс расшифрования производится по тому же алгоритму, что и шифрование. Единственное отличие заключается в порядке использования подключей Ki. При расшифровании подключи должны быть использованы в обратном порядке, а именно, как указано в таблице 2.

Таблица 2. Последовательность использования подключей при расшифровании

Раунд	1	2	3	4	5	6	7	8
Подключ	K0	K1	K2	K3	K4	K5	K6	K7
Раунд	9	10	11	12	13	14	15	16
Подключ	K7	K6	K5	K4	K3	K2	K1	K0
Раунд	17	18	19	20	21	22	23	24
Подключ	K7	K6	K5	K4	K3	K2	K1	K0
Раунд	25	26	27	28	29	30	31	32
Подключ	K7	K6	K5	K4	K3	K2	K1	K0

Режим простой замены: все блоки шифруются независимо друг от друга с разными подключами в разных раундах. Для одинаковых блоков сообщения M блоки шифртекста будут одинаковыми.

Режим гаммирования: В регистры N1 и N2 записывается 64-битовая синхропосылка (вектор инициализации) и шифруется с использованием СК. Результат подается на вход регистров и снова шифруется с использованием ключа. Получается «одноразовый блокнот».

В режиме гаммирования с обратной связью для заполнения регистров N1 и N2, начиная со 2-го блока, используется результат зашифрования предыдущего блока открытого текста на рисунке 4.

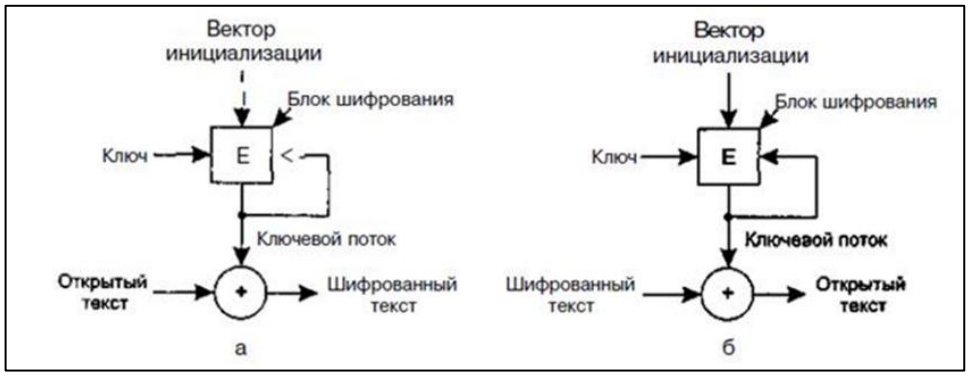

Рисунок 4 – Работа криптосистемы в режиме гаммирования

3 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 на рисунке 5.

```
PS C:\Workspace\BSUIR-Labs\MZI\Lab1> python.exe main.py encrypt in.txt out.txt
[PKCS7] +4 байт

Блок 00 (ENC) IN: 7465737404040404  'test....'
Раунд  1: N1=0x04040404, N2=0x0FC203EE
Раунд  2: N1=0x0FC203EE, N2=0xC7F8C388
Раунд  3: N1=0xC7F8C388, N2=0x6A439604
Раунд  4: N1=0x6A439604, N2=0xF4AD51A1
Раунд  5: N1=0xF4AD51A1, N2=0xB6AB7E9E
Раунд  6: N1=0xB6AB7E9E, N2=0x46FD92D5
Раунд  7: N1=0x46FD92D5, N2=0x5D9FDD7B
Раунд  8: N1=0x5D9FDD7B, N2=0xC7441887
Раунд  9: N1=0xC7441887, N2=0xA37ED630
Раунд 10: N1=0xA37ED630, N2=0x4E72694E
Раунд 11: N1=0x4E72694E, N2=0x54900D9E
Раунд 12: N1=0x54900D9E, N2=0x50CE1425
Раунд 13: N1=0x50CE1425, N2=0x41B638F0
Раунд 14: N1=0x41B638F0, N2=0xA09167D9
Раунд 15: N1=0xA09167D9, N2=0x0EF2D7F4
Раунд 16: N1=0x0EF2D7F4, N2=0xB134F163
Раунд 17: N1=0xB134F163, N2=0x771D50AC
Раунд 18: N1=0x771D50AC, N2=0x8D14635E
Раунд 19: N1=0x8D14635E, N2=0xEB9A89D1
Раунд 20: N1=0xEB9A89D1, N2=0xCF06C512
Раунд 21: N1=0xCF06C512, N2=0x2A88CC34
Раунд 22: N1=0x2A88CC34, N2=0xD58DA501
Раунд 23: N1=0xD58DA501, N2=0x44D57070
Раунд 24: N1=0x44D57070, N2=0xEAC4B86E
Раунд 25: N1=0xEAC4B86E, N2=0xA5F408DF
Раунд 26: N1=0xA5F408DF, N2=0xF7C8377C
Раунд 27: N1=0xF7C8377C, N2=0xBDF59861
Раунд 28: N1=0xBDF59861, N2=0x9CAD5C6E
Раунд 29: N1=0x9CAD5C6E, N2=0x7110E1E3
Раунд 30: N1=0x7110E1E3, N2=0x4E17F63B
Раунд 31: N1=0x4E17F63B, N2=0xF0A3AA46
Раунд 32: N1=0xF0A3AA46, N2=0x6524A857

Шифртекст (HEX) записан в: out.txt
```

Рисунок 5 – 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.

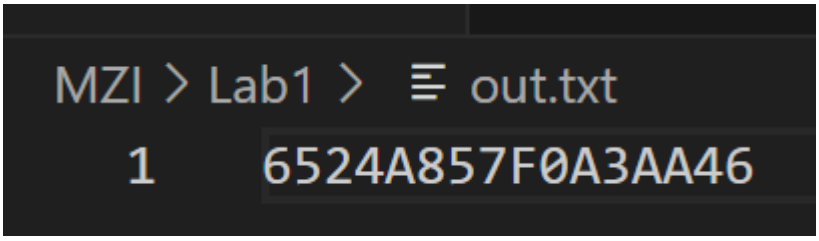

Рисунок 6 – Результат работы программы

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

KEY_HEX = "00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
16 17 18 19 1A 1B 1C 1D 1E 1F"
KEY = bytes.fromhex(KEY_HEX)

SBOX_HEX = """
04 0A 09 02 0D 08 00 0E 06 0B 01 0C 07 0F 05 03
0E 0B 04 0C 06 0D 0F 0A 02 03 08 01 00 07 05 09
05 08 01 0D 0A 03 04 02 0E 0F 0C 07 06 00 09 0B
07 0D 0A 01 00 08 09 0F 0E 04 06 0C 0B 02 05 03
06 0C 07 01 05 0F 0D 08 04 0A 09 0E 00 03 0B 02
04 0B 0A 00 07 02 01 0D 03 06 08 05 09 0C 0F 0E
0D 0B 04 01 03 0F 05 09 00 0A 0E 07 06 08 02 0C
01 0F 0D 00 05 07 0A 04 09 02 03 0E 06 0B 08 0C
""".strip()
SBOX_BYTES = bytes.fromhex(" ".join(SBOX_HEX.split()))


def rotl32(x, n):
    x &= 0xFFFFFFFF
    n &= 31
    return ((x << n) & 0xFFFFFFFF) | (x >> (32 - n))


def substitute(x):
    out = 0
    for i in range(8):
        nib = (x >> (4 * i)) & 0xF
        sb = SBOX_BYTES[i * 16 + nib] & 0xF
        out |= sb << (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"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"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
    print(f"[PKCS7] +{padlen} байт")
    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]
    print(f"[PKCS7] last=0x{p:02X}")
    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"\nБлок   {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"\nБлок {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"\nШифртекст (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"\nРасшифрованный текст записан в: {out_path}")


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