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 №2
на тему

СИММЕТРИЧНАЯ КРИПТОГРАФИЯ. СТБ 34.101.31-2011

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 обостряют потребность в надежной защите информации при хранении и передаче. Одним из ключевых инструментов обеспечения безопасности являются симметричные блочные шифры, которые позволяют гарантировать конфиденциальность и целостность данных, а также аутентификацию сообщений.

В данной лабораторной работе ставится задача изучить национальный стандарт симметричного шифрования СТБ 34.101.31-2011 и на его основе разработать программное средство, позволяющее выполнять шифрование и дешифрование текстовых файлов. Особенность работы заключается в том, что требуется не просто реализовать алгоритм как «чёрный ящик», но и отразить работу каждого раунда, что обеспечивает понимание внутренних механизмов построения блочных криптосистем.

Таким образом, цель работы можно сформулировать следующим образом: изучить теоретические основы стандарта СТБ 34.101.31-2011, реализовать программный алгоритм в режимах простой замены (ECB) и гаммирования с обратной связью (CFB-128)**,** протестировать его на примерах и убедиться в правильности выполнения операций шифрования и расшифрования.

Для достижения поставленной цели необходимо:

1 Рассмотреть структуру блочного шифра стандарта и порядок работы раундовых преобразований.

2 Описать процесс формирования подключей из 256-битового ключа.

3 Изучить роль таблицы подстановки (*H*-преобразование), обеспечивающей нелинейность.

4 Реализовать все основные стадии алгоритма: сложение и вычитание по модулю 2^{32}, циклические сдвиги, подстановки и обмен данными между словами.

5 Реализовать режим гаммирования с обратной связью (CFB-128), позволяющий обрабатывать сообщения произвольной длины с использованием синхропосылки (IV).

6 Создать интерфейс командной строки для запуска программы в режимах «encrypt» и «decrypt».

7 Провести эксперименты по шифрованию и дешифрованию файлов, отследить промежуточные шаги и зафиксировать результаты.

В работе рассматриваются два режима. Режим ECB является простым и наглядным: каждый блок шифруется независимо, однако одинаковые блоки открытого текста дают одинаковые блоки шифртекста, что снижает стойкость. Режим CFB-128 лишён этого недостатка и превращает блочный шифр в потоковый, обеспечивая защиту даже при наличии повторяющихся блоков. Использование обоих режимов позволяет на практике увидеть различия в их свойствах и наглядно проанализировать работу стандарта.

2 ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Настоящий стандарт определяет семейство криптографических алгоритмов, предназначенных для обеспечения конфиденциальности и контроля целостности данных. Обрабатываемыми данными являются двоичные слова (сообщения).

Криптографические алгоритмы стандарта построены на основе базовых алгоритмов шифрования блока данных.

Криптографические алгоритмы шифрования и контроля целостности делятся на восемь групп:

1) алгоритмы шифрования в режиме простой замены;
2) алгоритмы шифрования в режиме сцепления блоков;
3) алгоритмы шифрования в режиме гаммирования с обратной связью;
4) алгоритмы шифрования в режиме счетчика;
5) алгоритм выработки имитовставки;
6) алгоритмы одновременного шифрования и имитозащиты данных;
7) алгоритмы одновременного шифрования и имитозащиты ключа;
8) алгоритм хеширования.

Первые четыре группы предназначены для обеспечения конфиденциальности сообщений. Каждая группа включает алгоритм зашифрования и алгоритм расшифрования.

Стороны, располагающие общим ключом, могут организовать конфиденциальный обмен сообщениями путем их зашифрования перед отправкой и расшифрования после получения. В режимах простой замены и сцепления блоков шифруются сообщения, которые содержат хотя бы один блок, а в режимах гаммирования с обратной связью и счетчика – сообщения произвольной длины.

Пятый алгоритм предназначен для контроля целостности сообщений с помощью имитовставок – контрольных слов, которые определяются с использованием ключа. Стороны, располагающие общим ключом, могут организовать контроль целостности при обмене сообщениями путем добавления к ним имитовставок при отправке и проверки имитовставок при получении. Проверка имитовставок дополнительно позволяет стороне-получателю убедиться в том, что сторона-отправитель знает ключ, т. е. позволяет проверить подлинность сообщений.

Блок-схема алгоритма на i-ом такте шифрования представлена на рисунке 1

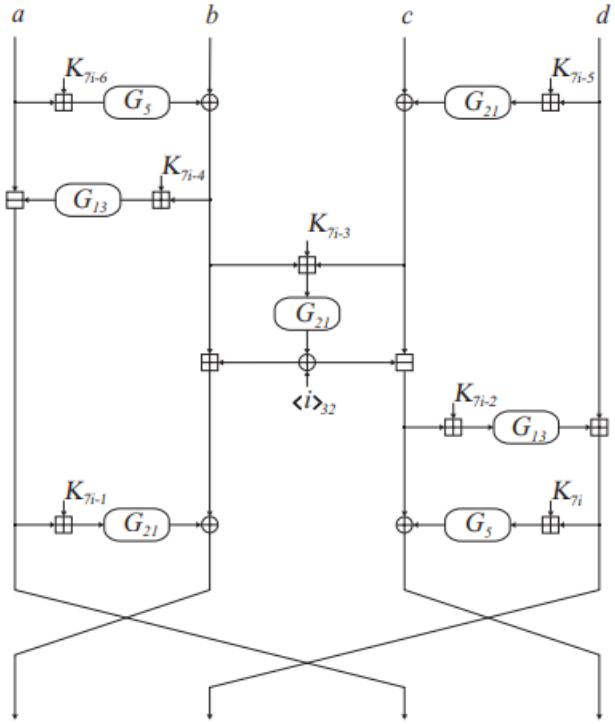

Рисунок 1 – Вычисления на i-м такте шифрования

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:

1 Слово X записывается в виде $X = X_1 \| X_2 \| X_3 \| X_4, X_i \in \{0,1\}^{32}$.

2 Ключ θ записывается в виде $\theta = \theta_1 \| \theta_2 \| \theta_3 \| \theta_4 \| \theta_5 \| \theta_6 \| \theta_7 \| \theta_8, \theta_i \in \{0,1\}^{32}$ и определяются тактовые ключи $K_1 = \theta_1, K_2 = \theta_2, K_3 = \theta_3, K_4 = \theta_4, K_5 = \theta_5, K_6 = \theta_6, K_7 = \theta_7, K_8 = \theta_8, K_9 = \theta_1, \ldots\ldots, K_{56} = \theta_8$.

Обозначения и вспомогательные преобразования

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

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

циклический сдвиг влево на r бит.

$H(u)$ операция замены 8-битной входной строки подстановкой с рисунка 2.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	B1	94	BA	C8	0A	08	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	B0	C0	FF	33	C3	56	B8	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	30	6A	DD	4E	A7	79	9E	B2	3D	31
7	3E	98	B5	6E	27	D3	BC	CF	59	1E	18	1F	4C	5A	B7	93
8	E9	DE	E7	2C	8F	0C	0F	A6	2D	DB	49	F4	6F	73	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	48	A0	2A	88	5F	19	4B	09	A1
D	7E	CD	A4	D0	15	44	AF	8C	A5	84	50	BF	66	D2	E8	8A
E	A2	D7	46	52	42	A8	DF	B3	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 2 – Преобразование H

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

Установить $a \leftarrow X_1, b \leftarrow X_2, c \leftarrow X_3, d \leftarrow X_4$.

Для i = 8,7,… ,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 $Y \leftarrow c \parallel a \parallel d \parallel b$.

4. Возвратить Y.

Входными данными алгоритмов зашифрования и расшифрования являются сообщение $X \in \{0, 1\}^*$, ключ O$\in\{0, 1\}^{256}$ и синхропосылка $S \in \{0, 1\}^{128}$.

Выходными данными является слово $Y \in \{0, 1\}^{|X|}$ — результат зашифрования либо расшифрования X на ключе θ при использовании синхропосылки S.

Входное сообщение X записывается в виде $X = X_1 \parallel X_2 \parallel \ldots \parallel X_n$, $|X_1| = |X_2| = \ldots = |X_{n-1}| = 128$, $|X_n| <= 128$.

При шифровании словам Xi ставятся в соответствие слова $Y_i \in \{0, 1\}^{|Xi|}$, из которых затем составляется Y .

При зашифровании используется вспомогательный блок $Y_0 \in \{0, 1\}^{128}$, а при расшифровании — вспомогательный блок $X0 \in \{0, 1\}^{128}$

Зашифрование сообщения X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1 Установить $Y_0 \leftarrow S$.

2 Для $i = 1, 2, \ldots, n$ выполнить: $Y_i \leftarrow X_i \oplus L|X_i|(F(Y_{i-1}))$.

3 Установить $Y \leftarrow Y_1 \parallel Y_2 \parallel \ldots \parallel Y_n$.

4 Возвратить Y.

Расшифрование сообщения X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1 Установить $X_0 \leftarrow S$.

2 Для $i = 1, 2, \ldots, n$ выполнить: $Y_i \leftarrow X_i \oplus L|X_i|(F\theta(X_{i-1}))$.

3 Установить $Y \leftarrow Y_1 \parallel Y_2 \parallel \ldots \parallel Y_n$.

4 Возвратить Y.

3 ХОД РАБОТЫ

Для выполнения лабораторной работы был реализован программный комплекс на языке Python, воспроизводящий работу алгоритма симметричного шифрования по стандарту СТБ 34.101.31-2011. В рамках реализации были предусмотрены два режима работы: простой замены (ECB) и гаммирования с обратной связью (CFB-128).

Алгоритм был реализован строго в соответствии с описанием стандарта. В режиме ECB данные делились на блоки длиной 128 бит, каждый блок проходил 8 раундов преобразований. Внутри каждого раунда выполнялись следующие операции:

1 Сложение и вычитание по модулю 2^{32}.

2 Нелинейное преобразование (*H*-преобразование), основанное на таблице подстановки.

3 Циклические сдвиги на фиксированное число бит.

4 Обмен значениями между словами блока.

Перед шифрованием текст дополнялся байтами по стандарту PKCS#7, что обеспечивало кратность длины блока 16 байтам. Результат шифрования сохранялся в выходной файл в двоичном виде. При расшифровании выполнялся обратный процесс: данные снова делились на блоки, проходили 8 раундов преобразований в обратном порядке, после чего снималась добивка, и текст возвращался в исходный вид.

В режиме CFB-128 был реализован потоковый принцип шифрования: для каждого блока вычислялась гамма как результат шифрования текущего значения обратной связи (IV или предыдущего шифртекста). Эта гамма накладывалась на открытый текст с помощью операции XOR. На этапе расшифрования использовался тот же процесс, но в качестве обратной связи выступали блоки шифртекста. Такой режим позволял обрабатывать данные произвольной длины, включая файлы, не кратные размеру блока.

Программа была снабжена отладочным выводом, позволяющим наблюдать внутренние состояния алгоритма рисунок 3.

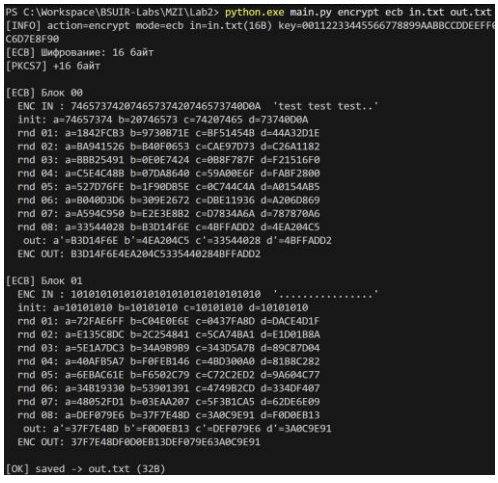

Рисунок 3 – Начало работы программы

В качестве примера тестирования использовалась строка «test test test». В процессе шифрования можно было наблюдать формирование блоков, добавление паддинга и последовательное изменение слов состояния на каждом раунде. При расшифровании отчётливо отразился блок добивки вида `0x04040404`, соответствующий корректному PKCS#7. После удаления добивки исходный текст был полностью восстановлен рисунок 4.

```
PS C:\Workspace\BSUIR-Labs\MZI\Lab2> python.exe main.py decrypt ecb out.txt in.txt
[INFO] action=decrypt mode=ecb in=out.txt(32B) key=00112233445566778899AABBCCDDEEF
5C6D7E8F90
[ECB] Расшифрование: 32 байт, блоков: 2

[ECB] Блок 00
  DEC IN : B3D14F6E4EA204C5335440284BFFADD2
  init(dec): a=B3D14F6E b=4EA204C5 c=33544028 d=4BFFADD2
  rnd 01: a=E2E3E8B2 b=787870A6 c=A594C950 d=D7834A6A
  rnd 02: a=309E2672 b=A206D869 c=B040D3D6 d=DBE11936
  rnd 03: a=1F90DB5E b=A0154AB5 c=527D76FE d=0C744C4A
  rnd 04: a=07DA8640 b=FABF2800 c=C5E4C48B d=59A00E6F
  rnd 05: a=0E0E7424 b=F21516F0 c=BBB25491 d=0B8F787F
  rnd 06: a=B40F0653 b=C26A1182 c=BA941526 d=CAE97D73
  rnd 07: a=9730B71E b=44A32D1E c=1842FCB3 d=BF51454B
  rnd 08: a=20746573 b=73740D0A c=74657374 d=74207465
   out(dec): a''=74657374 b''=20746573 c''=74207465 d''=73740D0A
  DEC OUT: 74657374207465737420746573740D0A  'test test test..'

[ECB] Блок 01
  DEC IN : 37F7E48DF0D0EB13DEF079E63A0C9E91
  init(dec): a=37F7E48D b=F0D0EB13 c=DEF079E6 d=3A0C9E91
  rnd 01: a=03EAA207 b=62DE6E09 c=48052FD1 d=5F3B1CA5
  rnd 02: a=53901391 b=334DF407 c=34B19330 d=4749B2CD
  rnd 03: a=F6502C79 b=9A604C77 c=6EBAC61E d=C72C2ED2
  rnd 04: a=F0FEB146 b=81B8C282 c=40AFB5A7 d=4BD300A0
  rnd 05: a=34A9B9B9 b=89C87D04 c=5E1A7DC3 d=343D5A7B
  rnd 06: a=2C254841 b=E1D01B8A c=E135C8DC d=5CA74BA1
  rnd 07: a=C04E0E6E b=DACE4D1F c=72FAE6FF d=0437FA8D
  rnd 08: a=10101010 b=10101010 c=10101010 d=10101010
   out(dec): a''=10101010 b''=10101010 c''=10101010 d''=10101010
  DEC OUT: 10101010101010101010101010101010  '................'

[ECB] Снятие PKCS#7…
[PKCS7] last=0x10

[OK] saved -> in.txt (16B)
```

Рисунок 4 – Результат работы программы

Таким образом, было подтверждено, что реализация алгоритма является корректной: процедуры шифрования и дешифрования полностью обратимы, а все промежуточные состояния соответствуют описанию алгоритма СТБ 34.101.31-2011.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была достигнута основная цель – реализовать на практике один из национальных стандартов симметричного шифрования СТБ 34.101.31-2011. В процессе изучения были рассмотрены основные элементы алгоритма: блочная структура, организация раундов, особенности формирования подключей из 256-битового ключа, а также роль H-преобразования в обеспечении нелинейности.

Отдельное внимание было уделено реализации двух режимов работы алгоритма: ECB и CFB-128. В режиме простой замены (ECB) продемонстрирована наглядность шифрования блоков фиксированной длины и работа механизма PKCS#7-паддинга. В режиме гаммирования с обратной связью (CFB-128) реализован потоковый принцип преобразования, позволяющий шифровать данные произвольной длины и устраняющий статистические закономерности, присущие режиму ECB.

Важным этапом стало то, что программа была создана не в виде «чёрного ящика», а с подробным отладочным выводом. Это позволило наблюдать промежуточные значения на каждом раунде: выполнение операций сложения по модулю 2^{32}, подстановки, циклического сдвига и обмена данными между словами блока. Такой подход способствовал более глубокому пониманию структуры алгоритма и его внутренней логики.

Практическая часть продемонстрировала корректность реализации. Исходные данные были зашифрованы и успешно восстановлены после дешифрования. Корректная работа паддинга PKCS#7 и совпадение исходного текста с результатом расшифрования подтвердили полную обратимость алгоритма и соответствие требованиям симметричных шифров.

Таким образом, все поставленные задачи были выполнены. Разработанное программное средство может использоваться как учебный инструмент для наглядного изучения принципов работы стандарта СТБ 34.101.31-2011 и блочных криптосистем в целом. Лабораторная работа позволила закрепить теоретические знания о криптографических методах защиты информации и продемонстрировать их практическую реализацию с использованием современного языка программирования Python.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
import sys, argparse, binascii

BLOCK = 16
KEY_HEX = "00112233445566778899AABBCCDDEEFF0F1E2D3C4B5A69788796A5B4C3D2E1F0"
IV_HEX = "A1B2C3D4E5F60718293A4B5C6D7E8F90"
KEY = binascii.unhexlify(KEY_HEX)
IV = binascii.unhexlify(IV_HEX)

H_HEX = (
    "B1 94 BA C8 0A 08 F5 3B 36 6D 00 8E 58 4A 5D E4"
    " 85 04 FA 9D 1B B6 C7 AC 25 2E 72 C2 02 FD CE 0D"
    " 5B E3 D6 12 17 B9 61 81 FE 67 86 AD 71 6B 89 0B"
    " 5C B0 C0 FF 33 C3 56 B8 35 C4 05 AE D8 E0 7F 99"
    " E1 2B DC 1A E2 82 57 EC 70 3F CC F0 95 EE 8D F1"
    " C1 AB 76 38 9F E6 78 CA F7 C6 F8 60 D5 BB 9C 4F"
    " F3 3C 65 7B 63 7C 30 6A DD 4E A7 79 9E B2 3D 31"
    " 3E 98 B5 6E 27 D3 BC CF 59 1E 18 1F 4C 5A B7 93"
    " E9 DE E7 2C 8F 0C 0F A6 2D DB 49 F4 6F 73 96 47"
    " 06 07 53 16 ED 24 7A 37 39 CB A3 83 03 A9 8B F6"
    " 92 BD 9B 1C E5 D1 41 01 54 45 FB C9 5E 4D 0E F2"
    " 68 20 80 AA 22 7D 64 2F 26 87 F9 34 90 40 55 11"
    " BE 32 97 13 43 FC 9A 48 A0 2A 88 5F 19 4B 09 A1"
    " 7E CD A4 D0 15 44 AF 8C A5 84 50 BF 66 D2 E8 8A"
    " A2 D7 46 52 42 A8 DF B3 69 74 C5 51 EB 23 29 21"
    " D4 EF D9 B4 3A 62 28 75 91 14 10 EA 77 6C DA 1D"
)
H_BYTES = binascii.unhexlify(H_HEX.replace(" ", ""))


def pkcs7_pad(b):
    n = BLOCK - (len(b) % BLOCK)
    if n == 0:
        n = BLOCK
    print(f"[PKCS7] +{n} байт")
    return b + bytes([n]) * n


def pkcs7_unpad(b):
    n = b[-1]
    print(f"[PKCS7] last=0x{n:02X}")
    if n < 1 or n > BLOCK or b[-n:] != bytes([n]) * n:
        raise ValueError("PKCS#7 error")
    return b[:-n]


def xor_bytes(a, b):
    return bytes(x ^ y for x, y in zip(a, b))


def _b2l16(b):
    return list(b)


def _l2b16(l):
    return bytes(l)


def _hex(b):
    return b.hex().upper()
```

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


class STB:
    def __init__(self, key_bytes):
        self.Hb = H_BYTES
        kw = [self._list_to_int(key_bytes[i : i + 4]) for i in range(0, 32, 4)]
        self.k = [kw[i % 8] for i in range(56)]

    def _int_to_list(self, x):
        return [(x >> i) & 0xFF for i in (24, 16, 8, 0)]

    def _list_to_int(self, xs):
        sh = (24, 16, 8, 0)
        return (xs[0] << sh[0]) | (xs[1] << sh[1]) | (xs[2] << sh[2]) | (xs[3]
<< sh[3])

    def _rev(self, x):
        l = self._int_to_list(x)
        l.reverse()
        return self._list_to_int(l)

    def _rotl32(self, v, k):
        k &= 31
        v &= 0xFFFFFFFF
        return ((v << k) & 0xFFFFFFFF) | (v >> (32 - k))

    def _modadd(self, *vals):
        s = 0
        for el in vals:
            s = (s + self._rev(el)) & 0xFFFFFFFF
        return self._rev(s)

    def _modsub(self, x, y):
        return (x - y) & 0xFFFFFFFF

    def _H(self, x):
        return self.Hb[x]

    def _G(self, x, r):
        t = self._list_to_int([self._H(b) for b in self._int_to_list(x)])
        return self._rev(self._rotl32(self._rev(t), r))

    def encryption(self, m, trace=False):
        a, b, c, d = [self._list_to_int(m[i : i + 4]) for i in range(0, 16, 4)]
        if trace:
            print(f"  init: a={a:08X} b={b:08X} c={c:08X} d={d:08X}")
        for i in range(8):
            b ^= self._G(self._modadd(a, self.k[7 * i + 0]), 5)
            c ^= self._G(self._modadd(d, self.k[7 * i + 1]), 21)
            a = self._rev(
                self._modsub(
                    self._rev(a),
                    self._rev(self._G(self._modadd(b, self.k[7 * i + 2]), 13)),
                )
            )
            e = (self._G(self._modadd(b, c, self.k[7 * i + 3]), 21)) ^
self._rev(i + 1)
            b = self._modadd(b, e)
            c = self._rev(self._modsub(self._rev(c), self._rev(e)))
```

```
            d = self._modadd(d, self._G(self._modadd(c, self.k[7 * i + 4]),
13))
            b ^= self._G(self._modadd(a, self.k[7 * i + 5]), 21)
            c ^= self._G(self._modadd(d, self.k[7 * i + 6]), 5)
            a, b = b, a
            c, d = d, c
            b, c = c, b
            if trace:
                print(f"    rnd  {i+1:02d}:  a={a:08X}  b={b:08X}  c={c:08X}
d={d:08X}")
        out = (
            self._int_to_list(b)
            + self._int_to_list(d)
            + self._int_to_list(a)
            + self._int_to_list(c)
        )
        if trace:
            oa, ob, oc, od = [
                self._list_to_int(out[i : i + 4]) for i in range(0, 16, 4)
            ]
            print(f"   out: a'={oa:08X} b'={ob:08X} c'={oc:08X} d'={od:08X}")
        return out

    def decryption(self, m, trace=False):
        a, b, c, d = [self._list_to_int(m[i : i + 4]) for i in range(0, 16, 4)]
        if trace:
            print(f"  init(dec): a={a:08X} b={b:08X} c={c:08X} d={d:08X}")
        for i in reversed(range(8)):
            b ^= self._G(self._modadd(a, self.k[7 * i + 6]), 5)
            c ^= self._G(self._modadd(d, self.k[7 * i + 5]), 21)
            a = self._rev(
                self._modsub(
                    self._rev(a),
                    self._rev(self._G(self._modadd(b, self.k[7 * i + 4]), 13)),
                )
            )
            e = (self._G(self._modadd(b, c, self.k[7 * i + 3]), 21)) ^
self._rev(i + 1)
            b = self._modadd(b, e)
            c = self._rev(self._modsub(self._rev(c), self._rev(e)))
            d = self._modadd(d, self._G(self._modadd(c, self.k[7 * i + 2]),
13))
            b ^= self._G(self._modadd(a, self.k[7 * i + 1]), 21)
            c ^= self._G(self._modadd(d, self.k[7 * i + 0]), 5)
            a, b = b, a
            c, d = d, c
            a, d = d, a
            if trace:
                print(f"    rnd  {8-i:02d}:  a={a:08X}  b={b:08X}  c={c:08X}
d={d:08X}")
        out = (
            self._int_to_list(c)
            + self._int_to_list(a)
            + self._int_to_list(d)
            + self._int_to_list(b)
        )
        if trace:
            oa, ob, oc, od = [
                self._list_to_int(out[i : i + 4]) for i in range(0, 16, 4)
            ]
            print(f"      out(dec):  a''={oa:08X}  b''={ob:08X}  c''={oc:08X}
d''={od:08X}")
        return out
```

```
def enc_block(b):
    stb = STB(KEY)
    print(f"  ENC IN : { _hex(b) }  '{ _ascii(b) }'")
    out = stb.encryption(_b2l16(b), trace=True)
    outb = _l2b16(out)
    print(f"  ENC OUT: { _hex(outb) }")
    return outb


def dec_block(b):
    stb = STB(KEY)
    print(f"  DEC IN : { _hex(b) }")
    out = stb.decryption(_b2l16(b), trace=True)
    outb = _l2b16(out)
    print(f"  DEC OUT: { _hex(outb) }  '{ _ascii(outb) }'")
    return outb


def ecb_encrypt(data):
    print(f"[ECB] Шифрование: {len(data)} байт")
    data = pkcs7_pad(data)
    out = bytearray()
    for i in range(0, len(data), BLOCK):
        print(f"\n[ECB] Блок {i//BLOCK:02d}")
        out += enc_block(data[i : i + BLOCK])
    return bytes(out)


def ecb_decrypt(ct):
    if len(ct) % BLOCK != 0:
        raise ValueError("ECB: длина не кратна 16")
    print(f"[ECB] Расшифрование: {len(ct)} байт, блоков: {len(ct)//BLOCK}")
    out = bytearray()
    for i in range(0, len(ct), BLOCK):
        print(f"\n[ECB] Блок {i//BLOCK:02d}")
        out += dec_block(ct[i : i + BLOCK])
    print("\n[ECB] Снятие PKCS#7…")
    return pkcs7_unpad(bytes(out))


def cfb_encrypt(data, iv):
    print(f"[CFB] Шифрование: {len(data)} байт, IV={IV_HEX}")
    out = bytearray()
    fb = iv
    for i in range(0, len(data), BLOCK):
        print(f"\n[CFB-ENC] Блок {i//BLOCK:02d}")
        ks = enc_block(fb)
        chunk = data[i : i + BLOCK]
        y = xor_bytes(chunk, ks[: len(chunk)])
        print(f"   KS : { _hex(ks[:len(chunk)]) }")
        print(f"   IN : { _hex(chunk) }  '{ _ascii(chunk) }'")
        print(f"   OUT: { _hex(y) }")
        out += y
        fb = y if len(y) == 16 else (y + fb[len(y) :])
    return bytes(out)


def cfb_decrypt(ct, iv):
    print(f"[CFB] Расшифрование: {len(ct)} байт, IV={IV_HEX}")
    out = bytearray()
    fb = iv
    for i in range(0, len(ct), BLOCK):
        print(f"\n[CFB-DEC] Блок {i//BLOCK:02d}")
```

```
        ks = enc_block(fb)
        chunk = ct[i : i + BLOCK]
        x = xor_bytes(chunk, ks[: len(chunk)])
        print(f"   KS : { _hex(ks[:len(chunk)]) }")
        print(f"   IN : { _hex(chunk) }")
        print(f"   OUT: { _hex(x) }  '{ _ascii(x) }'")
        out += x
        fb = chunk if len(chunk) == 16 else (chunk + fb[len(chunk) :])
    return bytes(out)


def main():
    p = argparse.ArgumentParser(
        description="СТБ 34.101.31-2011: ECB/CFB (хардкод, трейс)"
    )
    p.add_argument("action", choices=["encrypt", "decrypt"])
    p.add_argument("mode", choices=["ecb", "cfb"])
    p.add_argument("inp")
    p.add_argument("out")
    args = p.parse_args()

    with open(args.inp, "rb") as f:
        data = f.read()
    print(
        f"[INFO]             action={args.action}            mode={args.mode}
in={args.inp}({len(data)}B) key={KEY_HEX} iv={IV_HEX}"
    )

    if args.mode == "ecb":
        res   =   ecb_encrypt(data)   if   args.action   ==   "encrypt"   else
ecb_decrypt(data)
    else:
        res = (
            cfb_encrypt(data,   IV)   if   args.action   ==   "encrypt"   else
cfb_decrypt(data, IV)
        )

    with open(args.out, "wb") as f:
        f.write(res)
    print(f"\n[OK] saved -> {args.out} ({len(res)}B)")


if __name__ == "__main__":
    if len(sys.argv) < 5:
        print("Использование: python main.py encrypt(decrypt) ecb(cfb) in.txt
out.txt")
        sys.exit(2)
    main()
```