1.Алгоритм RSA

```
z=1;
for i =s-1 downto 0 do
z= z^2 mod n
if c_i =1 then z=z*x mod n
```

7591^2437mod1123 = **1039**

c = 2437
x = 7591
n = 1123

$(2437)_{10} = (100110000101)_2 = Ci$

i	c_i	Z = 1
11	1	**z= z² mod n** = 1^2 mod 1123 = 1* 7591 mod 1123 = 7591
10	0	**z= z² mod n** = 7591^2 mod 1123 = 1028
9	0	**z= z² mod n** = 1028^2 mod 1123 =41
8	1	**z= z² mod n** = 41^2 mod 1123 = 558 * 7591 mod 1123 = 945
7	1	**z= z² mod n** = 945^2 mod 1123 = 240 * 7591 mod 1123 = 334
6	0	**z= z² mod n** =334^2 mod 1123 = 379
5	0	**z= z² mod n** = 379^2 mod 1123 = 1020
4	0	**z= z² mod n** =1020^2 mod 1123 = 502
3	0	**z= z² mod n** = 502^2 mod 1123 = 452
2	1	**z= z² mod n** = 452^2 mod 1123 = 1041* 7591 mod 1123 = 803
1	0	**z= z² mod n** =803^2 mod 1123 = 207
0	1	**z= z² mod n** =207^2 mod 1123 = 175 *7591 mod 1123 = 1039

2.Calculul inversei modulo

1. $n_0 = n$; $b_0 = b$; $t_0 = 0$; $t = 1$;

2.$q = n_0 / b_0$; $r = n_0 – q*b_0$;

3.while r > 0 **do**

3.1.temp = $t_0 – q*t$;

3.2.**if** temp >= 0 **then** temp = temp **mod** n

else temp = n – ((- temp) **mod** n)

3.3$n_0 = b_0$; $b_0 = r$; $t_0 = t$; t= temp;

3.4.$q = [n_0/ b_0]$; $r = n_0 – q*b_0$;

4.if b_0 != 1 **then** b nu are inversă **mod** n

else b^{-1} **mod** n = t;

1103^{-1} mod 2137= 31?

b = 1103

n = 2137

	n_0	b_0	q	r	t_0	t	temp
1	2137	1103	1	1034	0	1	2136
2	1103	1034	1	69	1	2136	2
3	1034	69	14	68	2136	2	2108
4	69	68	1	1	2	2108	31
5	68	1	68	0	2108	31	0
6	1	0	беск		31	0	

Calcule matematice:

1. $n_0 = 2137$; $b_0 = 1103$; $t_0 = 0$; $t = 1$; $q = [2137/ 1103]= 1$; $r = 2137 – 1*1103 = 1034$; **temp = 0 – 1*1 = - 1;**

temp = 2137 – ((-(- 1)) **mod** 2137) = 2137– 1 = 2136 mod 2137= 2136

2. n_0 = b0=1103; b_0 = r=1034; t_0 = 1; t= 2136; q = [1103/ 1034] =1; r = 1103– 1*1034 =69; temp = 1 – 1*2136 = -2135;

temp = 2137– ((-(- 2135)) **mod** 2137) = 2137– (2135 mod 2137) = 2;

3. n_0 = 1034; b_0 = 69; t_0 = 2136; t= 2; q = 1034/ 69= 14; r = 1034– 14*69 = 68; temp = 2136– 14*2 = 2108;

4. n_0 = 69; b_0 = 68; t_0 = 2; t= 2108; q = [69/ 68]=1 ; *r = 69 – 1*68 = 1*; temp = 2 – 1*2108 = -2106; temp = 2137– ((-(-2106)) **mod** 2137) = 31;

5. n_0 = 68; ***b_0 = 1***; t_0 = 2108; **t= 31**; q = 68/ 1 = 68; r = 68 – 68*1 =0; **R!>0** temp = 2108– 68*31 = 0;

6. n_0 = 1; b_0 =0; t_0 = 31; t= 0; q = 1/ 0 = беск;

Răspuns: b^{-1} **mod** n = t ; 1103^{-1} mod 2137= 31

1109^{-1} mod 3037= 953?

b = 1109

n = 3037

	n_0	b_0	q	r	t_0	t	temp

1	3037	1109	2	819	0	1	3035
2	1109	819	1	290	1	3035	3
3	819	290	2	239	3035	3	3029
4	290	239	1	51	3	3029	11
5	239	51	4	35	3029	11	2985
6	51	35	1	16	11	2985	63
7	35	16	2	3	2985	63	2859
8	16	3	5	1	63	2859	953
9	3	1	3	0	2859	953	0

Calcule matematice:

1. n_0 = 3037; b_0 = 1109; t_0 = 0; t = 1; q = [3037/ 1109]= 2; r = 3037– 2*1109 = 819; **temp = 0 – 2*1 = - 2;**

temp = 3037– ((-(- 2)) **mod** 3037) = 3037– 2 = 3035 mod 3037=3035

2. n_0 =b0=1109; b_0 = r=819; t_0 = 1; t= 3035; q = [1109/ 819] =1; r = 1109– 1*819 = 290; temp = 1 – 1*3035 = -3034;

temp = 3037– ((-(- 3034)) **mod** 3035) = 3037– (3034 mod 3035) = 3;

3. n_0 = 819; b_0 = 290; t_0 = 3035; t= 3; q = 819/ 290= 2; r = 819– 2*290 = 239; temp = 3035– 2*3 = 3029;

4. n_0 = 290; b_0 = 239; t_0 = 3; t=3029; q = [290/ 239]=1 ; r = 290– 1*239 = 51; temp = 3 – 1*3029 = -3026; temp = 3037– ((-(-3026)) **mod** 3037) = 3037– (3026 mod 3037) = 11;

5. n_0 = 239; b_0 = 51; t_0 = 3029; t= 11; q = 239/ 51 = 4; r = 239– 4*51 = 35; temp = 3029– 4*11 = 2985;

6. n_0 = 51; b_0 = 35; t_0 = 11; t= 2985; q = 51/ 35= 1; r = 51 – 1*35 = **16**; temp = 11– 1*2985= - 2974; temp = 3037– ((-(-2974)) **mod** 3037) = 3037– (2974 mod 3037) = 63

7. n_0 = 35; b_0 = 16; t_0 = 2985; t=63; q = 35/ 16= 2; r = 35– 2*16 = 3; temp = 2985– 2*63= 2859;

8. n_0 = 16; b_0 = 3; t_0 = 63; t=2859; q = 16/3 = 5; r = 16– 5*3 = 1; temp = 63– 5*2859=-14232;

temp = 3037– ((-(-14232)) **mod** 3037) = 3037– (14232 mod 3037) = 953

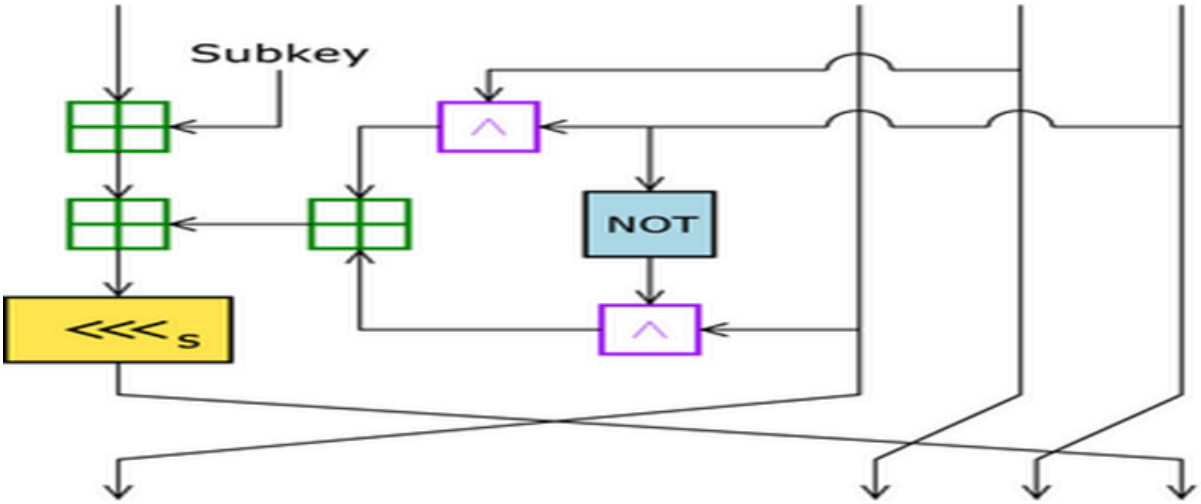
9. n_0 = 3; b_0 = 1; t_0 = 2859; **t=953**; q = 3/ 1= 3; r = 3– 3*1 = 0; temp = 2859– 3*953= 0;

Răspuns: b^{-1} **mod** n = t ; 1109^{-1} mod 3037= 953

3. Realizare Algoritmul RIVEST CODE 2

Сообщение шифрования m = “fantasia” в бинарном виде : 01100110 01100001 01101110 01110100 01100001 01110011 01101001 01100001.

Subkey –CS.



a => (01100110 01100001**), b => (**01101110 01110100**)**

c => (01100001 01110011**), d => (**01101001 01100001**)**

Subkey CS=> (01000011 01010011)

Первый шаг : a + subkey => (01100110 01100001 + **01000011 01010011**) mod 2^{32} = (26209+**17235**) mod 2^{32} = (43444 mod 2^{32} = 43444=> **10101001 10110100**

Второй шаг : c^d = 01100001 01110011$^{01101011\ 01100001}$ = 01100001 01100001

Третий шаг : not$(d)^b$ = not(01101001 01100001)$^{01101110\ 01110100}$ = 10010110 10011110$^{01101110\ 01110100}$ = 00000110 00010100

Четвертый шаг : (c^d + not$(d)^b$) mod 2^{32} = (01100001 01100001 + 00000110 00010100) mod 2^{32} = (24929 + 1556) mod 2^{32} = 26485 mod 2^{32} = 26485=> 01100111 01110101

Пятый шаг : ((a + subkey) + (c^d + not$(d)^b$)) mod 2^{32} = (**10101001 10110100** + 01100111 01110101) mod 2^{32} = (43444 + 26485) mod 2^{32} = 69929=> **10001000100101001**

Шестой шаг : Переместить биты с пятого шага влево на 4 (<<<4) = 00000000 0000000**10001000100101001**(добавляем нули слева, чтобы расширить на 32 бит, тк получили в пятом пункте не 16 бит, а 17) => 00000000 000**10001000100101001**0000

Седьмой шаг : b => (01101110 01110100), c => (01100001 01110011)

d => (01101001 01100001), a => (00000000 000**10001 00010010 1001**0000)

Восьмой шаг : Конкатенация (склеивание) b, c, d и a = (01101110 01110100 01100001 01110011 01101001 01100001 00000000 000**10001 00010010 1001**0000) =>

Result = "ntasia" + "????"

Расшифровка

Первый шаг : Переместить биты с шестого шага шифровки вправо на 4 (>>>4) = 00000000 000**10001 00010010 1001**0000 => 0000000000000000**10001 00010010 1001**

Второй шаг : c^d = 01100001 01110011$^{01101011\ 01100001}$ = 01100001 01100001

Третий шаг : not(d)^b = not(01101001 01100001)$^{01101110\ 01110100}$ = 10010110 10011110$^{01101110\ 01110100}$ = 00000110 00010100

Четвертый шаг : (c^d + not(d)^b) mod 2^{32} = (01100001 01100001 + 00000110 00010100) mod 2^{32} = (24929 + 1556) mod 2^{32} = 26485 mod 2^{32} = 26485=> 01100111 01110101

Пятый шаг : (a - (c^d + not(d)^b)) mod 2^{32} = (0000000000000000**10001 000011101001** - 01100111 00110101) mod 2^{32} = (69865 - 26421) mod 2^{32} = 43444=> **1010100110110100**

Шестой шаг : ((a – CS) mod 2^{32} = (**1010100110110100** - **01000011 01010011**) mod 2^{32} = (43444 – 17235) mod 2^{32} =26209=> **110011001100001**

Седьмой шаг : Конкатенация (склеивание) a, b, c и d = (0**1100110 01100001** 01101110 01110100 01100001 01110011 01101001 01100001) => “fantasia”

Result = "fantasia"

4. Digital signature with RSA (первых 4х букв имени)

1. Генерация ключей RSA:

Модуль n:
n=p×q=53×109=5777

Функция Эйлера ϕ(n):
ϕ(n)=(p−1)×(q−1)=(53−1)×(109−1)=52×108=5616

Выбираем открытый ключ e=17 (меньшее простое число, взаимно простое с φ(n)):
Теперь проверим gcd(e,φ(n))=1, что верно, поскольку 17 и 5616 взаимно просты.

Вычисление закрытого ключа d:
d должно удовлетворять уравнению d×e≡1 mod ϕ(n)d. Используя расширенный алгоритм Евклида, находим d. В данном случае d=3305.

2. Шифрование имени "SOFI":

Используем ASCII-кодировку:

- **"S" = 83**
- **"O" = 79**
- **"F" = 70**
- **"I" = 73**

Теперь шифруем каждую букву отдельно, используя формулу **c = m^e mod n,** где e=17 и n=5777**.**

Для "**S**" (83): c=83^17 mod 5777=3363

Для "**O**" (79): c=79^17 mod 5777=4196

Для "**F**" (70): c=70^17 mod 5777=5300

Для "**I**" (73): c=73^17 mod 5777=1811

зашифрованное сообщение для "SOFI" будет: 3363 4196 5300 1811.

3. Дешифрование:

Теперь дешифруем сообщение, используя формулу m=c^d mod n, где d=3305 и n=5777.

Для 3363: m=3363^3305 mod 5777=83 (буква "S")

Для 4196: m=4196^3305 mod 5777=79 (буква "O")

Для 5300: m=5300^3305mod 5777=70 (буква "F")

Для 1811: m=1811^3305mod 5777=73 (буква "I")

В результате мы получаем исходные символы: "SOFI".

Лабораторная работа №1
Алгоритм шифрования SAFER

История и Разработка

SAFER (Secure And Fast Encryption Routine) — это семейство блочных шифров, разработанное Джеймсом Л. Мэсси в 1993 году. Основная цель заключалась в создании алгоритма, который бы обеспечивал высокую скорость выполнения при достаточной криптографической стойкости. Название SAFER получило несколько модификаций, таких как SAFER K-64, SAFER SK и SAFER+, каждая из которых улучшала устойчивость к криптоанализу и поддерживала ключи разной длины.

SAFER был популярен в 1990-е и 2000-е годы благодаря простоте реализации и скорости работы, особенно в встраиваемых системах и беспроводной связи. Например, версия SAFER+ была предложена для использования в стандарте Bluetooth.

Принципы работы

SAFER — это симметричный блочный шифр, обрабатывающий данные блоками фиксированного размера (64 или 128 бит). Шифрование и дешифрование основаны на применении нескольких итераций нелинейных и линейных операций к блокам данных, делая их максимально непохожими на исходный текст.

Основные характеристики:

- **Тип шифра**: блочный.
- **Размер блока**: 64 бита (SAFER K-64) или 128 бит (SAFER+).
- **Размер ключа**: 64, 128 или 256 бит.
- **Число итераций**: от 6 до 10 в зависимости от версии.

Шифрование SAFER включает несколько основных шагов, которые повторяются в каждой итерации. В разных версиях алгоритма используется разное количество итераций, а ключи могут быть разной длины.

Основные этапы шифрования

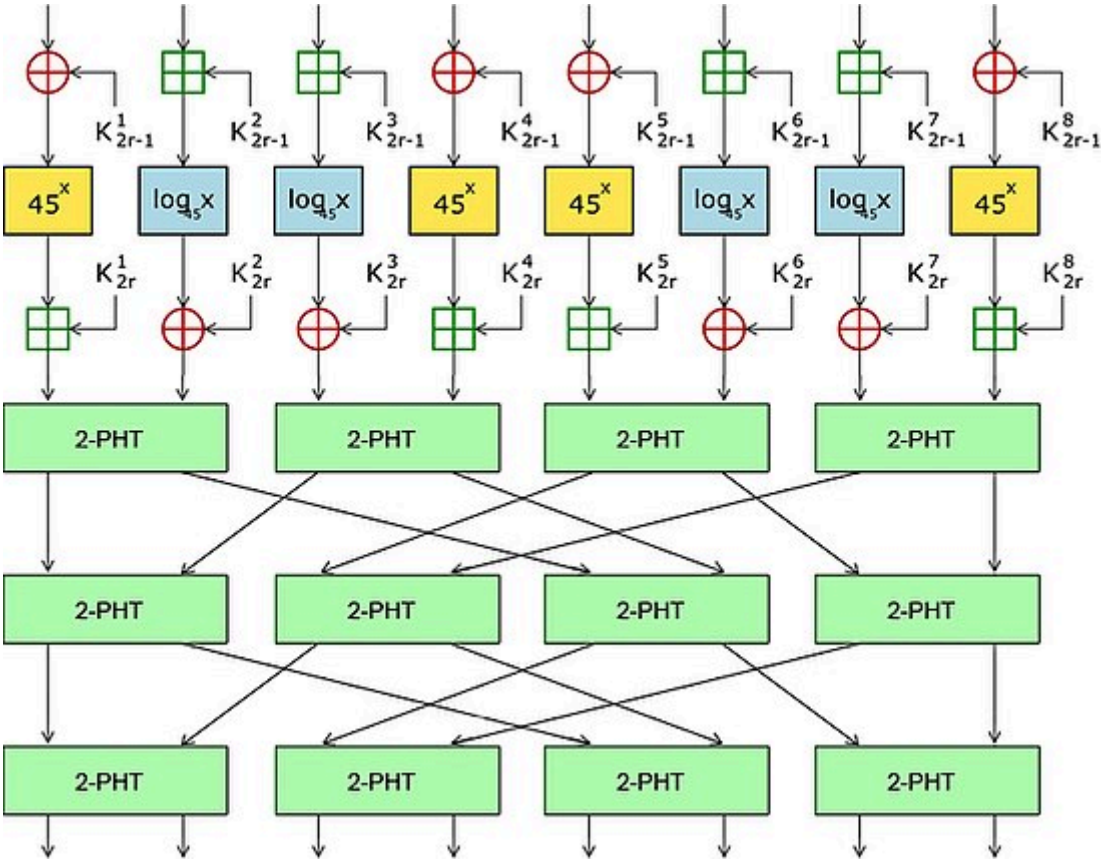
1. **Разделение на блоки**: Сообщение разбивается на блоки фиксированного размера, например, по 64 бита (в версии SAFER K-64).
2. **Генерация итерационных ключей**: Исходный ключ преобразуется в набор ключей, которые будут использоваться на каждой итерации шифрования. В алгоритме SAFER K-64 ключ расширяется для 16 итераций, а в SAFER+ — для 10.
3. **Итерационные преобразования**:
    - **XOR с ключом**: На каждом шаге блок данных обрабатывается с использованием итерационного ключа с применением операции XOR.
    - **Нелинейные преобразования**: Применяются S-блоки (замены) для создания нелинейности и усложнения анализа зашифрованных данных.
    - **Циклические сдвиги**: Данные подвергаются циклическому сдвигу, чтобы усилить перемешивание битов между итерациями.
4. **Финальное преобразование**: После всех итераций выполняется заключительное преобразование с ключом, что завершает процесс шифрования.
5. **Дешифрование**: Процесс дешифрования зеркален шифрованию. Ключи используются в обратном порядке, а операции выполняются в обратной последовательности.

Преимущества и особенности

1. **Устойчивость к дифференциальному криптоанализу**: SAFER был спроектирован таким образом, чтобы защищать данные от атак, основанных на дифференциальном анализе, что являлось критически важным в 1990-е годы.
2. **Высокая скорость и легкость реализации**: SAFER был разработан для быстрой реализации как в программном обеспечении, так и в аппаратных решениях. Это делает его привлекательным для устройств с ограниченными вычислительными ресурсами.
3. **Гибкость ключей**: Возможность использования ключей различной длины позволяет настраивать уровень безопасности в зависимости от потребностей системы.

Применение

1. **Bluetooth**: SAFER+ был предложен в качестве одного из вариантов шифрования в стандарте Bluetooth, так как он предлагал хороший баланс между скоростью выполнения и безопасностью.
2. **Смарт-карты и встраиваемые системы**: Алгоритм широко использовался в системах, где требовалось шифрование с минимальными затратами ресурсов, например, в смарт-картах и устройствах с ограниченной мощностью процессора.



Для начала нужно инсталлировать библиотеку

```
pip install cryptography
```

Реализация на языке программирования Python

```
import random


# SAFER parameters (basic version)
BLOCK_SIZE = 8  # Block size in bytes
KEY_SIZE = 16   # Key size in bytes
NUM_ROUNDS = 8  # Number of rounds (can be varied)


# Generate a random key for encryption
def generate_key():
   return bytes([random.randint(0, 255) for _ in
range(KEY_SIZE)])


# Simple substitution box (S-box) for SAFER-like encryption
def sbox(byte):
   return (byte + 0x55) & 0xFF  # Simple substitution
function


# Reverse S-box for decryption
def inv_sbox(byte):
   return (byte - 0x55) & 0xFF


# SAFER-like encryption of one block
def encrypt_block(block, key):
   block = bytearray(block)
   for round in range(NUM_ROUNDS):
```

```
        for i in range(BLOCK_SIZE):
            block[i] ^= key[i % KEY_SIZE]  # XOR with key
            block[i] = sbox(block[i])      # S-box
substitution
    return bytes(block)


# SAFER-like decryption of one block
def decrypt_block(block, key):
    block = bytearray(block)
    for round in range(NUM_ROUNDS):
        for i in range(BLOCK_SIZE):
            block[i] = inv_sbox(block[i])  # Reverse S-box
            block[i] ^= key[i % KEY_SIZE]  # XOR with key
    return bytes(block)


# Pad the message to be a multiple of block size
def pad_message(message):
    pad_len = BLOCK_SIZE - (len(message) % BLOCK_SIZE)
    return message + bytes([pad_len] * pad_len)


# Remove padding after decryption
def unpad_message(padded_message):
    pad_len = padded_message[-1]
    return padded_message[:-pad_len]
```

```python
# SAFER encryption of a message
def encrypt_message(message, key):
    padded_message = pad_message(message)
    encrypted_message = b""
    for i in range(0, len(padded_message), BLOCK_SIZE):
        block = padded_message[i:i+BLOCK_SIZE]
        encrypted_block = encrypt_block(block, key)
        encrypted_message += encrypted_block
    return encrypted_message


# SAFER decryption of a message
def decrypt_message(encrypted_message, key):
    decrypted_message = b""
    for i in range(0, len(encrypted_message), BLOCK_SIZE):
        block = encrypted_message[i:i+BLOCK_SIZE]
        decrypted_block = decrypt_block(block, key)
        decrypted_message += decrypted_block
    return unpad_message(decrypted_message)


# Example usage
if __name__ == "__main__":
    key = generate_key()
    message = b"Secret message"  # Input message as bytes
    print(f"Original message: {message}")
```

```
    encrypted = encrypt_message(message, key)
    print(f"Encrypted message: {encrypted}")

    decrypted = decrypt_message(encrypted, key)
    print(f"Decrypted message: {decrypted}")
```

1. Параметры алгоритма SAFER:

```
BLOCK_SIZE = 8  # Размер блока в байтах
KEY_SIZE = 16   # Размер ключа в байтах
NUM_ROUNDS = 8  # Количество раундов (можно изменять)
```

- **BLOCK_SIZE**: Размер блока данных, который будет шифроваться за один шаг. В данном случае блок равен 8 байтам (64 бита).
- **KEY_SIZE**: Размер ключа шифрования. Здесь выбран 16-байтный (128-битный) ключ.
- **NUM_ROUNDS**: Количество раундов шифрования, через которые проходит каждый блок. Больше раундов означает более сильную криптографическую защиту, но также увеличивает время выполнения.

2. Генерация ключа:

```
def generate_key():
    return bytes([random.randint(0, 255) for _ in
range(KEY_SIZE)])
```

- Эта функция генерирует случайный симметричный ключ. Она возвращает 16-байтное значение (так как `KEY_SIZE = 16`).
- Каждый байт ключа — это случайное целое число в диапазоне от 0 до 255. Ключ представляет собой последовательность байтов (тип данных `bytes`).

3. S-box и обратная S-box:

```
def sbox(byte):
    return (byte + 0x55) & 0xFF  # Простая функция подстановки


# Обратная подстановка для расшифровки
def inv_sbox(byte):
    return (byte - 0x55) & 0xFF
```

- **S-box** (от англ. "substitution box") — это функция подстановки, которая принимает входной байт и изменяет его определенным образом. В нашем случае это простое сложение с постоянным значением `0x55` (85 в десятичной системе). Операция выполняется по модулю 256 (это достигается с помощью `& 0xFF`), чтобы результат всегда был байтом (от 0 до 255).
- **inv_sbox** — обратная подстановка, которая вычитает 85, восстанавливая исходное значение байта. Это необходимо для расшифровки.

Подобные подстановки используются для того, чтобы сделать шифрование нечувствительным к линейным зависимостям между входными и выходными данными.

4. Шифрование одного блока:

```
def encrypt_block(block, key):
    block = bytearray(block)
```

```
    for round in range(NUM_ROUNDS):
        for i in range(BLOCK_SIZE):
            block[i] ^= key[i % KEY_SIZE]  # XOR с ключом
            block[i] = sbox(block[i])      # Преобразование
S-box
    return bytes(block)
```

- **block** — это часть исходного сообщения длиной 8 байт, которая шифруется за одну итерацию.
- **key** — 16-байтный ключ, который применяется к каждому блоку.
- На каждой итерации:
    - Каждый байт блока подвергается побитовой операции XOR с байтом ключа. Поскольку размер ключа больше, чем размер блока, для каждого байта блока выбирается соответствующий байт ключа с помощью выражения `key[i % KEY_SIZE]`.
    - После этого к каждому байту блока применяется функция подстановки S-box.
- Процесс повторяется для заданного количества итераций (`NUM_ROUNDS`). Чем больше итераций, тем сложнее и надёжнее шифрование.

5. Расшифровка одного блока:

```
def decrypt_block(block, key):
    block = bytearray(block)
    for round in range(NUM_ROUNDS):
        for i in range(BLOCK_SIZE):
            block[i] = inv_sbox(block[i])  # Обратная S-box
            block[i] ^= key[i % KEY_SIZE]  # XOR с ключом
    return bytes(block)
```

- Сначала мы применяем обратную S-box функцию к каждому байту блока, затем выполняем операцию XOR с байтом ключа, восстанавливая исходные данные.
- Порядок операций полностью противоположен шифрованию.

6. Функции добавления и удаления заполнения:

```
def pad_message(message):
    pad_len = BLOCK_SIZE - (len(message) % BLOCK_SIZE)
    return message + bytes([pad_len] * pad_len)
```

```
def unpad_message(padded_message):
    pad_len = padded_message[-1]
    return padded_message[:-pad_len]
```

- функция **pad_message**: Добавляет (padding) к сообщению, чтобы его длина стала кратной размеру блока (`BLOCK_SIZE`). Если длина сообщения уже кратна 8 байтам, всё равно добавляется дополнительный блок для избежания двусмысленности при расшифровке.
    - `pad_len` — это количество байтов, которое необходимо добавить. Например, если сообщение длиной 10 байт, то для достижения 16 байт (следующего кратного значения) нужно добавить 6 байтов.
    - В конце сообщения добавляются байты, содержащие значение `pad_len`. Это позволяет потом легко удалить заполнение.

- **unpad_message**: Удаляет добавленное заполнение после расшифровки. Значение последнего байта (`padded_message[-1]`) указывает, сколько байтов заполнения нужно убрать.

7. Шифрование и расшифровка сообщения:

```
def encrypt_message(message, key):
    padded_message = pad_message(message)
    encrypted_message = b""
    for i in range(0, len(padded_message), BLOCK_SIZE):
        block = padded_message[i:i+BLOCK_SIZE]
        encrypted_block = encrypt_block(block, key)
        encrypted_message += encrypted_block
    return encrypted_message
```

```
def decrypt_message(encrypted_message, key):
    decrypted_message = b""
    for i in range(0, len(encrypted_message), BLOCK_SIZE):
        block = encrypted_message[i:i+BLOCK_SIZE]
        decrypted_block = decrypt_block(block, key)
        decrypted_message += decrypted_block
    return unpad_message(decrypted_message)
```

- **encrypt_message**:
    - Принимает исходное сообщение и ключ для шифрования.

    - Сообщение сначала дополняется заполнением (padding), чтобы его длина стала кратной 8 байтам.
    - Затем сообщение разбивается на блоки по 8 байт, и каждый блок шифруется с помощью функции `encrypt_block`.
    - Все зашифрованные блоки объединяются в одно зашифрованное сообщение.
- **decrypt_message**:
    - Принимает зашифрованное сообщение и ключ для расшифровки.
    - Сообщение также разбивается на блоки по 8 байт, и каждый блок расшифровывается с помощью функции `decrypt_block`.
    - После расшифровки удаляется заполнение с помощью функции `unpad_message`.

8. Пример шифрования сообщения:

```
key = generate_key()  # Генерация случайного ключа

message = b"secret message to encrypt"  # Сообщение в виде байтов

print(f"Оригинальное сообщение: {message}")


encrypted = encrypt_message(message, key)

print(f"Зашифрованное сообщение: {encrypted}")


decrypted = decrypt_message(encrypted, key)

print(f"Расшифрованное сообщение: {decrypted}")
```

- Здесь происходит генерация случайного ключа и зашифровка сообщения `"secret message to encrypt"`.
- После зашифровки выводится зашифрованное сообщение.

- Затем программа расшифровывает сообщение обратно и выводит его на экран, чтобы убедиться, что оно совпадает с исходным.

```
"C:\Users\kalin\Desktop\criptografia 4 octomber\.venv\Scripts\python.exe" "C:\Users\kalin\Desktop\criptografia 4 octombe
Оригинальное сообщение: b'secret message to encrypt'
Зашифрованное сообщение: b'\x13\xe5\x8bbe\xc4\xd0e\x053\x9bage\xd0|\x8f\xe0\x8d^#b\x19x\xb4\x87/\x07\x07\x07\xa7\xff'
Расшифрованное сообщение: b'secret message to encrypt'

Process finished with exit code 0
```

Заключение

SAFER представляет собой важный этап в развитии блочных шифров, особенно для тех случаев, где требуется баланс между скоростью и безопасностью. Несмотря на его меньшую популярность в наше время, алгоритм оказал значительное влияние на развитие современных стандартов шифрования, особенно в области беспроводной связи и встраиваемых систем.