

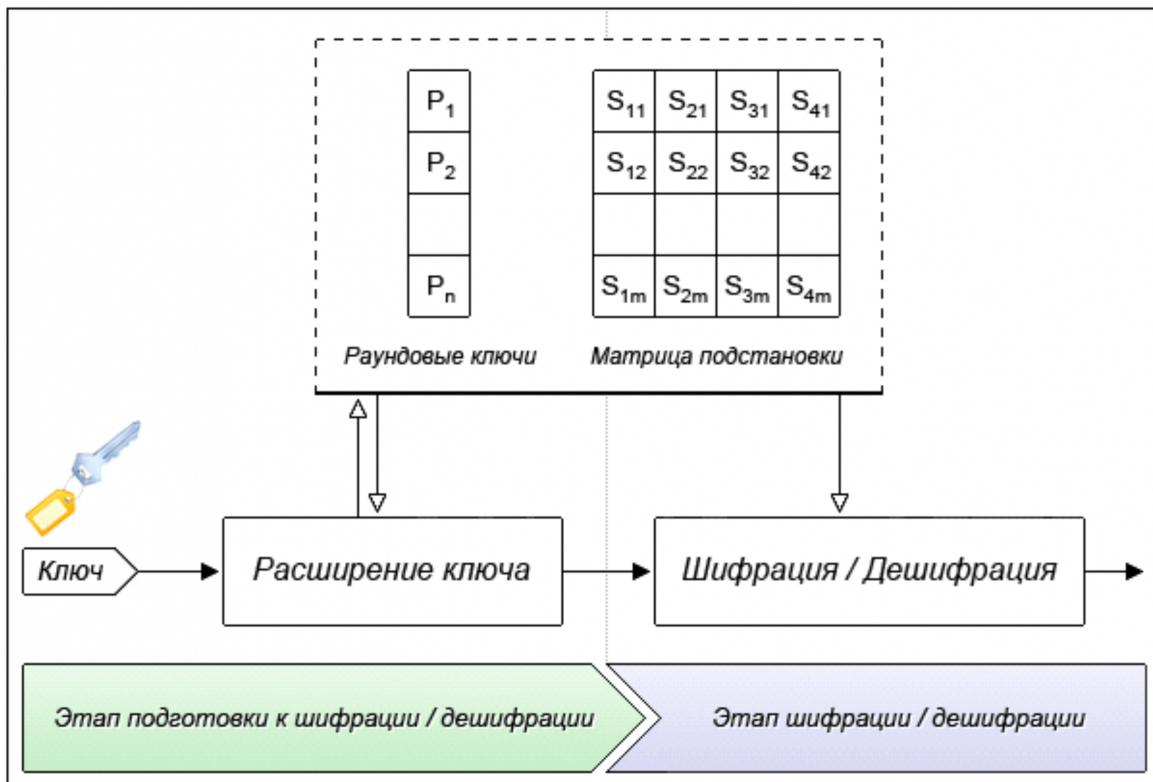
Защита информации. Лабораторная работа №4. Шифрование алгоритмом Blowfish

В 1994 году, на семинаре Fast Software Encryption в Кембридже, а впоследствии в журнале «Lecture Notes in Computer Science» (#809, 1994), Брюс Шнайер презентовал свой алгоритм блочного шифра, который был назван Blowfish.

Отличительными особенностями этого алгоритма стала более высокая степень криптостойкости, нежели алгоритма DES (в том числе за счет использования переменной длины ключа, до 448 бит), высокая скорость шифрации/десифрации (за счет генерации таблиц замены) и конечно — возможность его свободного применения для любых целей.

BlowFish — алгоритм 64-битного блочного шифра с ключом переменной длины. Был разработан известным специалистом в области криптографии и защиты информации Брюсом Шнайером (Bruce Schneier) в 1993 году.

В общем случае алгоритм состоит из двух этапов — расширение ключа и шифрация/десифрация исходных данных.



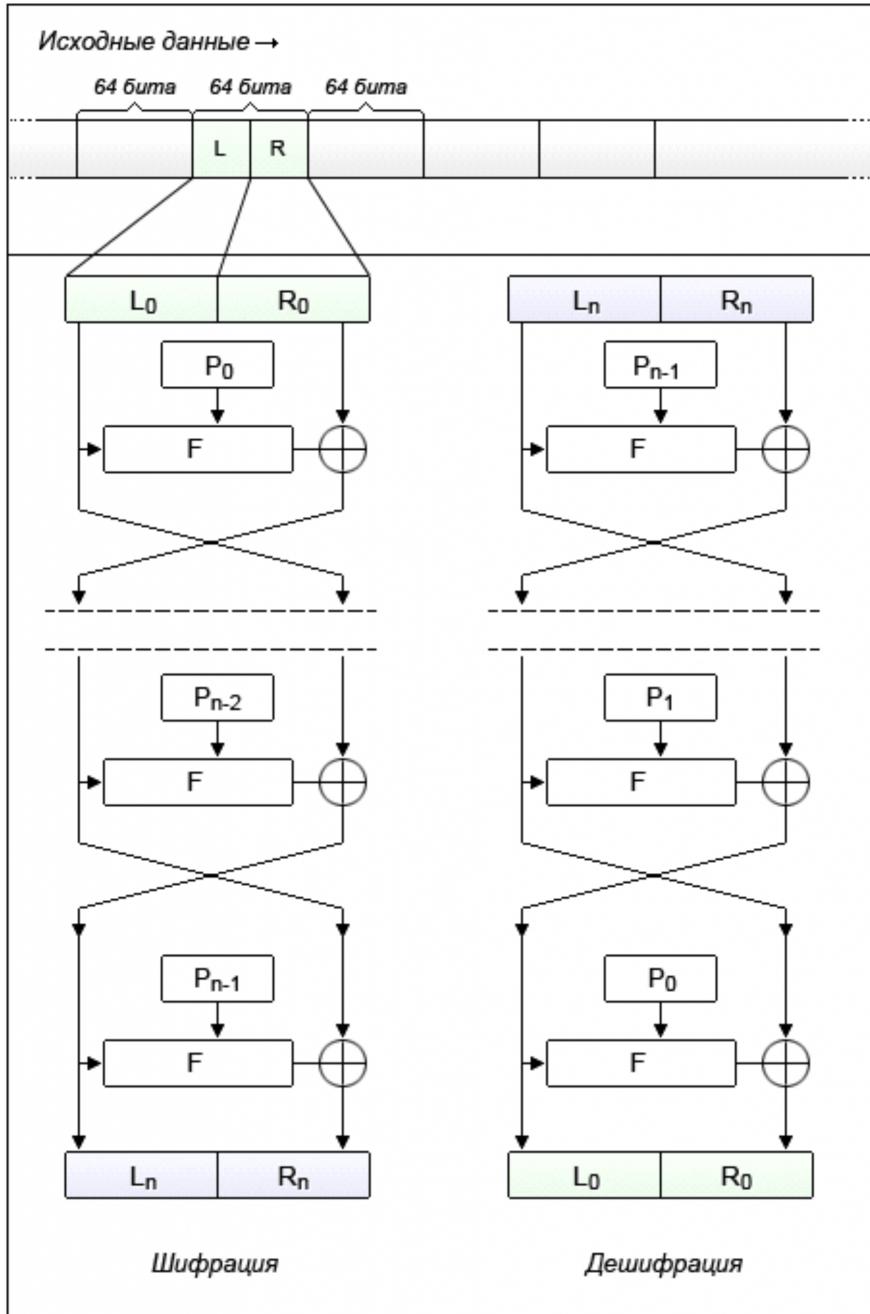
На этапе расширения ключа, исходный ключ преобразуется в матрицу раундовых ключей (P)

и матрицу подстановки (S, Substitution-box) (или замены), общим объемом в 4168 байт. По всей вероятности, этим «расширением» (от 448 бит до 4168 байт) и объясняется выбор названия алгоритма Blowfish.

Шифрация данных, а также создания матрицы раундовых ключей и подстановки, происходит через использование сети Фейстеля, состоящей в свою очередь из 16 раундов. Поэтому, перед тем, как рассмотреть этапы расширения ключа и шифрации данных в деталях, нам необходимо определиться, что из себя представляет упомянутая сеть Фейстеля.

Сеть Фейстеля

В 1971 году, «крестный отец» стандарта DES, Хорст Фейстель (Horst Feistel), в стенах корпорации IBM, разработал два устройства, реализовавшие различные алгоритмы шифрования, названные затем общим название «Люцифер». В одной из этих устройств он использовал схему, которую впоследствии назвали Сетью Фейстеля. Эта сеть представляет собой определённую многократно итерированную (повторяющуюся) структуру, которую называют ячейкой Фейстеля.



Принцип работы сети достаточно прост:

- Исходные данные разбиваются на блоки фиксированной длины (как правило кратно степени двойки — 64 бит, 128 бит). В случае если длина блока исходных данных меньше длины разрядности шифра, то блок дополняется каким-либо заранее известным образом.
- Блок делится на два равных подблока — «левый» L_0 и «правый» R_0 . В случае 64-битной разрядности — на два блока с длиной 32 бита каждый.

- «Левый подблок» L_0 видоизменяется функцией итерации $F(L_0, P_0)$ в зависимости от ключа P_0 ,
после чего он складывается по модулю 2 (XOR) с «правым подблоком» R_0 .
- Результат сложения присваивается новому левому подблоку L_1 , который становится левой половиной входных данных для следующего раунда, а «левый подблок» L_0 присваивается без изменений новому правому подблоку R_1 , который становится правой половиной.
- Эта операция повторяется $n-1$ раз, при этом при переходе от одного этапа к другому меняются раундовые ключи (P_0, P_1, P_2 и т.д.), где n — количество раундов для используемого алгоритма.

Процесс расшифрования аналогичен процессу шифрования за исключением того, что раундовые ключи используются в обратном порядке.

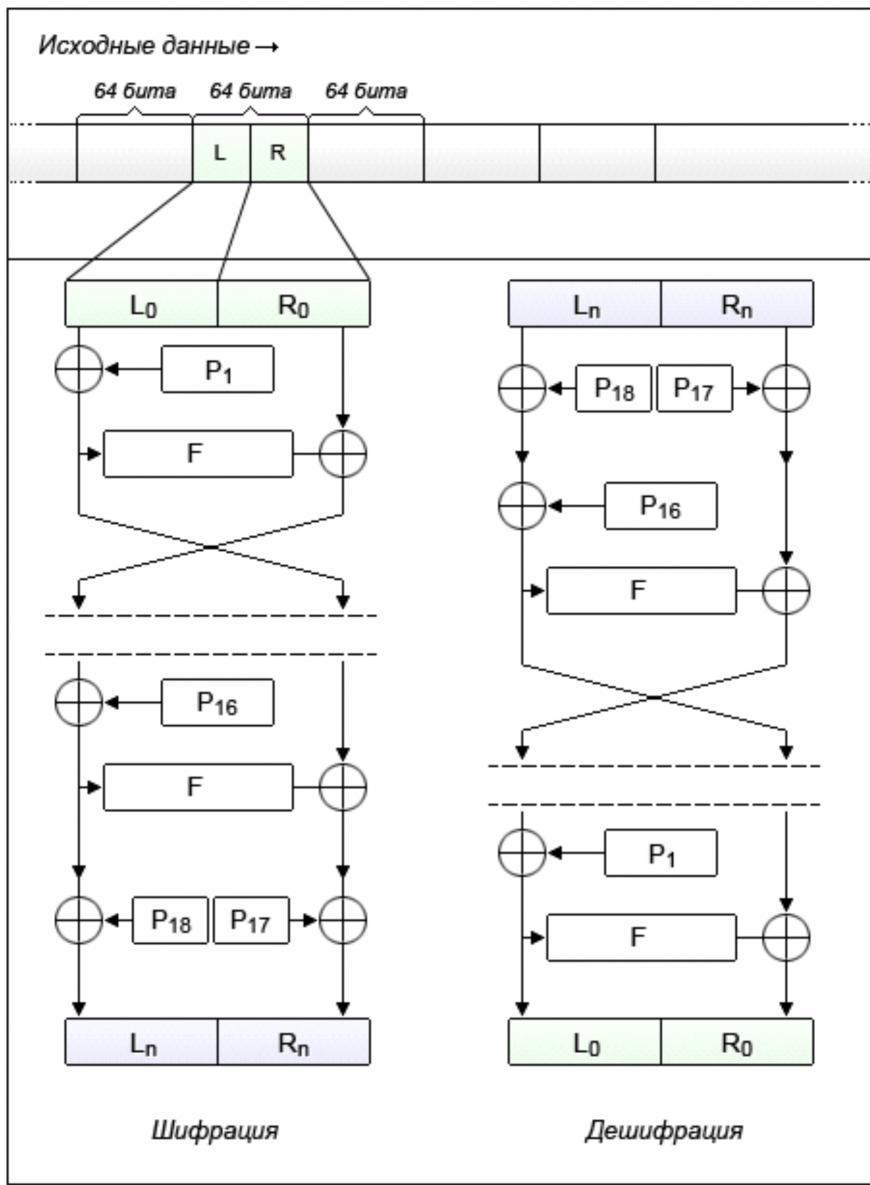
Вернемся к алгоритму Blowfish.

В общем случае, алгоритм шифрования Blowfish представляет собой сеть Фейстеля, но с некоторыми особенностями генерации и использования раундовых ключей ($P_0, P_1 \dots$).

Для начала допустим, что функция итерации F в алгоритме Blowfish это некоторый «черный ящик», который принимает на входе и выдает на выходе 32-битное число (DWORD).

При этом 32-битные раундовые ключи P_n :

- вычисляются по некоторому правилу от исходного ключа (длиной до 448 бит);
- не являются аргументами для функции итерации F ;
- непосредственно складываются по модулю 2 (XOR) с «левым блоком».
Результат этой операции является входящим 32-битным аргументом для функции F .



В алгоритме Blowfish при шифрации выполняется 16 раундов (внутри сети Фейстеля), а 17-й и 18-й ключи складываются с левым и правым выходным блоком последнего раунда. Такое количество раундов было выбрано, поскольку именно оно определяет длину возможного ключа.

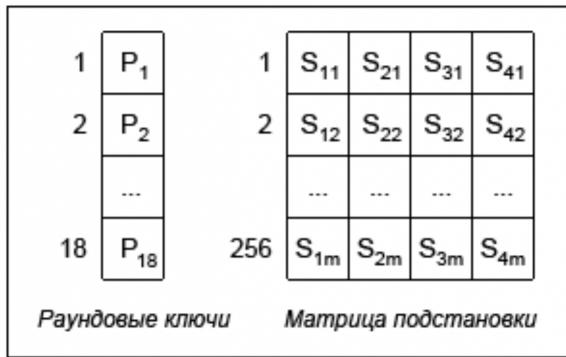
Но здесь у внимательного читателя может возникнуть вопрос: если используется 18 раундовых ключей, каждый из которых имеет длину 32 бита, то в итоге мы получаем ключ длиной 576 бит ($18 \text{ ключей} \times 32 \text{ бита}$). Почему же длина исходного ключа в Blowfish изначально ограничена 448 битами?

Ответ прост — она не ограничена. Можно использовать ключи до 576 бит. Но! Ограничение было сделано исходя из требований к соблюдению безопасности и

криптостойкости алгоритма.

Расширение ключа (Blow it up!)

Подготовительным этапом алгоритма Blowfish является этап расширения ключей. В процессе этого этапа строится матрица раундовых ключей P_n и матрица подстановки — 4 блока замены S-Box (Substitution-box), каждый из которых состоит из 256 32-х битных элементов.



Элементы этих матриц шифруются (вычисляются) с помощью рассмотренной выше сети Фейстеля для алгоритма Blowfish. Таким образом, сеть Фейстеля в алгоритме Blowfish используется:

- для шифрации/десифрации исходных данных;
- для генерации матрицы раундовых ключей и матрицы подстановки (т.е. расширения ключа).

Сгенерированная матрица раундовых ключей и матрицы подстановки впоследствии используются в процессе шифрации/десифрации.

В процессе расширения ключа обрабатывается $18 \times 32 (P_1, P_2, \dots) + 4 \times 256 \times 32 (S_1 - S_4) = 33344$ бит или 4168 байт данных.

При этом выполняется $(18 (P_n) + 4 \times 256 (S_1 - S_4)) / 2 = 521$ полная итерация сети Фейстеля.

Всё это является весьма трудозатратной операцией, и именно поэтому алгоритм Blowfish не рекомендуется использовать там, где требуется частая смена ключа.

Опишем алгоритм расширения ключа.

- Выбирается «искреннее число» (или иначе — «Nothing up my sleeve number»). Это такое число, которое изначально не содержит каких-либо повторяющихся

последовательностей и является известным. Делается это для того, чтобы показать, что константа разработчиками была выбрана без преследования каких-то «гнусных» целей, например для создания лазейки в алгоритме (backdoor).

В качестве такого искреннего числа в Blowfish обычно используется число PI. Вычислять шестнадцатеричное представление значение числа PI мы не будем, а воспользуемся уже готовым решением: [8366 шестнадцатеричных цифр мантиссы для числа PI](#).

- Значением мантиссы числа PI заполняется матрица раундовых ключей (FIXED_P) и матрицы подстановки (FIXED_S).

```
typedef struct _blowfish_ctx
{
    unsigned long p[18];
    unsigned long sbox[4][256];
} blowfish_ctx;

const unsigned int FIXED_S[4][256] = {
{
    0xD1310BA6, 0x98DFB5AC, 0x2FFD72DB, 0xD01ADFB7,
    0xB8E1AFED, 0x6A267E96, 0xBA7C9045, 0xF12C7F99,
    0x24A19947, 0xB3916CF7, 0x0801F2E2, 0x858EFC16,
    0x636920D8, 0x71574E69, 0xA458FEA3, 0xF4933D7E,
    0x0D95748F, 0x728EB658, 0x718BCD58, 0x82154AEE,
    0x7B54A41D, 0xC25A59B5, 0x9C30D539, 0x2AF26013,
    .... Часть значений не указаны ради экономии места ....
    0xB74E6132, 0xCE77E25B, 0x578FDFE3, 0x3AC372E6
};

const unsigned long FIXED_P[] = {
    0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344,
    0xA4093822, 0x299F31D0, 0x082EFA98, 0xEC4E6C89,
    0x452821E6, 0x38D01377, 0xBE5466CF, 0x34E90C6C,
    0xC0AC29B7, 0xC97C50DD, 0x3F84D5B5, 0xB5470917,
    0x9216D5D9, 0x8979FB1B
};
```

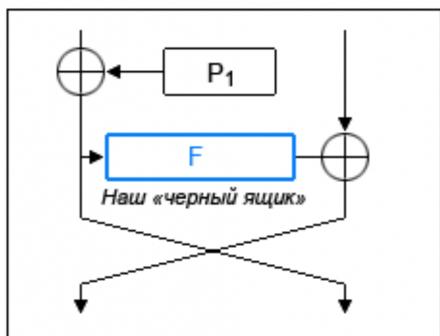
- Значение каждого раундового ключа P_n ($P_1, P_2 \dots$) складывается по модулю 2 (XOR) с соответствующими элементами исходного ключа K. Например, выполняется XOR раундового ключа P_1 с первыми 32 битами исходного ключа K, P_2 со вторыми 32 битами исходного ключа K и так далее. Если исходный ключ K короче длины всех раундовых ключей (576 бит), то он конкatenируется сам с собой: KK, KKK и так далее.
- Далее нам необходимо зашифровать (вычислить новые значения) элементов матрицы раундовых ключей и матрицы подстановки. Для этого мы воспользуемся реализованным нами алгоритмом сети Фейстеля для Blowfish.
 - Используя текущие раундовые ключи $P_1—P_{18}$ и матрицы подстановок $S_1—S_4$ (о том, где именно используются матрицы подстановок будет рассказано

ниже), шифруем 64-битную последовательность нуля: 0x00000000 0x00000000, а результат записываем в P_1 и P_2 .

- P_1 и P_2 шифруются изменёнными значениями раундовых ключей и матриц подстановки, результат записывается соответственно в P_3 и P_4 .
- Шифрование продолжается до изменения всех раундовых ключей $P_1—P_{18}$ и элементов матриц подстановок $S_1—S_4$.

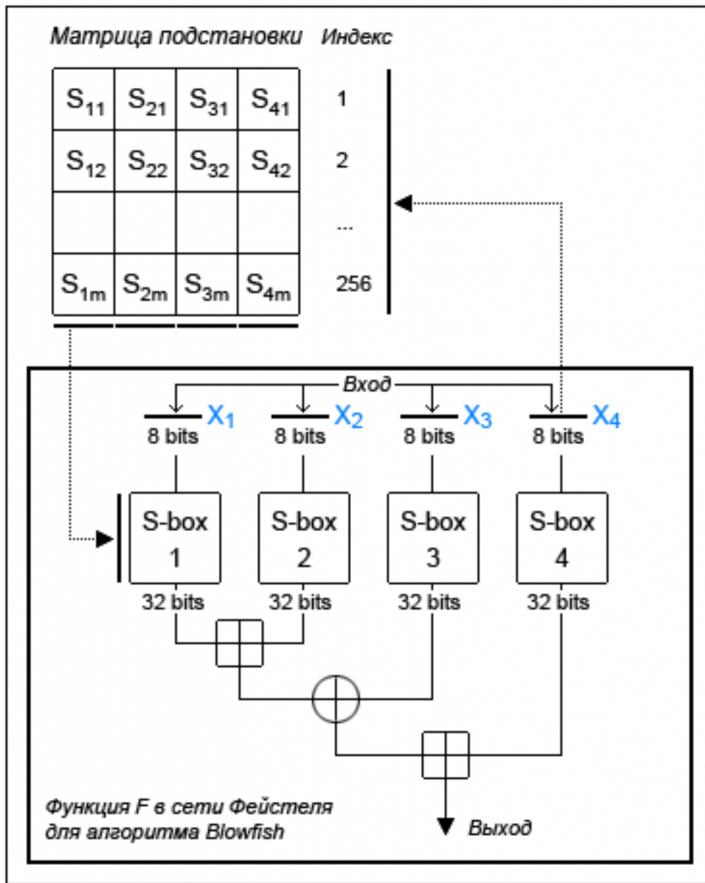
Т.е. в конечном счете нам надо получить результат вычисления по схеме сети Фейстеля алгоритма Blowfish для $(18 P_n + 4 \times 256 (S_1—S_4)) / 2 = 521$ итераций. Мы поделили на 2, поскольку за каждую итерацию мы вычисляем сразу два новых значения для элементов матрицы раундовых ключей или матрицы подстановок.

На этом, подготовительный этап алгоритма Blowfish — расширение ключа — завершается. Но перед тем, как рассмотреть этап шифрации/десифрации, давайте все-таки раскроем наш «чёрный ящик» — функцию F, исполняемую на каждом раунде внутри итераций сети Фейстеля для алгоритма Blowfish.



Функция итерации (раунда)

Функция раунда или итерации (поскольку она общая для всей используемой сети Фейстеля) весьма проста и использует лишь несколько логических операций над матрицей подстановок. Изначально, разрабатывая «Люцифера», Хорст Фейстель даже предполагал использование для матрицы подстановок электронного блока с простой линейной схемой.



Итак:

- Входящий 32-битный блок делится на четыре 8-битных блока, назовем их X_1, X_2, X_3, X_4 (см. рисунок выше).
- Каждый из которых является индексом массива таблицы замен $S_1—S_4$.
- значения $S_1[X_1]$ и $S_2[X_2]$ складываются по модулю 2^{32} , затем результат складывается по модулю 2 (XOR) с $S_3[X_3]$ и, наконец, складываются с $S_4[X_4]$ опять же по модулю 2^{32} .
- Результат вычислений и будет значением функции $F(X_1—X_4)$.

Формула функции:

$$F(X_1, X_2, X_3, X_4) = (((S_1[X_1] + S_2[X_2]) \bmod 2^{32} \oplus S_3[X_3]) + S_4[X_4]) \bmod 2^{32}$$

Всё крайне просто. Функция использует матрицы подстановок $S_1—S_4$ для того, чтобы линейно преобразовать входящие 32 бита данных в значение из матрицы подстановки. А

сами значения

в матрицах подстановки вычисляются на рассмотренном нами ранее этапе расширения ключа.

Реализация функции на языке C++:

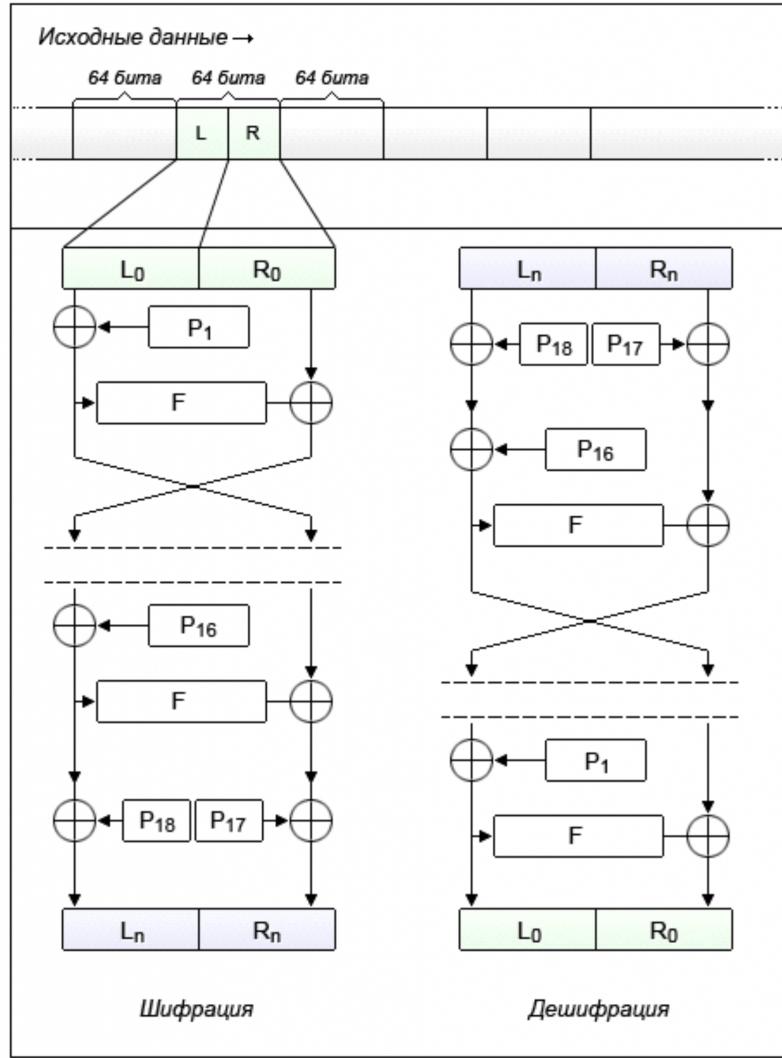
```
unsigned long F(blowfish_ctx *ctx, unsigned long x)
{
    return ((ctx->sbox[0][(x >> 24) & 0xFF] + ctx->sbox[1][(x >> 16) & 0xFF]) ^
            ctx->sbox[2][(x >> 8) & 0xFF]) + ctx->sbox[3][(x) & 0xFF];
}
```

А теперь давайте перейдем собственно к процессу шифрации исходных данных.

Шифрация / дешифрация исходных данных

А теперь самое интересное. На самом деле алгоритм шифрации и дешифрации исходных данных мы уже рассмотрели. Все дело в том, что как мы заметили в самом начале, шифрация / дешифрация данных, и создание вышеупомянутой матрицы раундовых ключей и матрицы подстановки, происходит с помощью рассмотренной сети Фейстеля для алгоритма Blowfish.

Т.е. в нашей программной реализации, весь процесс шифрации исходных данных строится по аналогии с процессом шифрации на этапе расширения ключа. Т.е. представляет собой итеративное выполнение функции `blowfish_encrypt_block` (реализация сети Фейстеля) над каждыми 64 битами исходных данных. Раундовые ключи P (P_1, P_2, \dots) и матрицы подстановки $S_1—S_4$ являются входными параметрами соответственно для сети Фейстеля и функции F внутри этой сети.



**В итоге, если резюмировать алгоритм шифрации или дешифрации в алгоритме Blowfish,
то получим следующие шаги:**

- Выделяем массив из 18 элементов для раундовых ключей сети Фейстеля и 4 матриц подстановки по 256 элементов в каждой.
- Заполняем выделенный массив значением мантиссы числа PI.
- Делаем итеративный XOR: $P_i = P_i \text{ XOR } K_i$ (где P_i — раундовый ключ, а K_i — исходный ключ).
- Шифруем раундовые ключи и матрицы подстановки с помощью сети Фейстеля (в качестве входящего параметра для функции внутри сети используются матрица подстановки; раундовые ключи внутри сети берутся из матрицы раундовых ключей).

- Шифруем/десифруем блоки исходных данных по 64 бита также с помощью сети Фейстеля.

Задание

Написать программу на любом языке программирования, реализующую шифрование и десифрование Blowfish. Проверить и обосновать правильность шифрования.

Найти ошибки в представлении сети Фейстеля (если таковые имеются).