**Темы задач к КР-2**

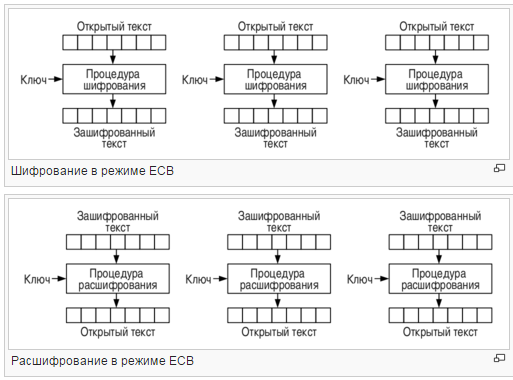
1. Описать и отобразить графически схему работы электронной кодовой книги

Самый простой режим работы назван режимом электронной кодовой книги (ECB — ELECTRONIC CODEBOOK). Исходный текст разделен на N блоков. Размер блока — n бит. Этот размер исходного текста не является кратным числом размера блока, текст дополняется, чтобы сделать последний блок по размеру таким же, как другие блоки. Один и тот же ключ используется, чтобы зашифровать и расшифровывать каждый блок.

Соотношение между исходным и зашифрованным текстами:

Шифрование: Ci = EK(Pi)

Дешифрование: Pi = DK(Ci)



1. Описать и отобразить графически алгоритм СВС

*CBC*) — один из [режимов шифрования](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B6%D0%B8%D0%BC_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) для симметричного [блочного шифра](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80) с использованием механизма обратной связи. Каждый блок открытого текста (кроме первого) побитово складывается по модулю 2 (операция [XOR](https://ru.wikipedia.org/wiki/XOR)) с предыдущим результатом шифрования.

Шифрование может быть описано следующим образом:

C_0=IV

C_i = E_k \left( P_{i} \oplus C_{i-1} \right)

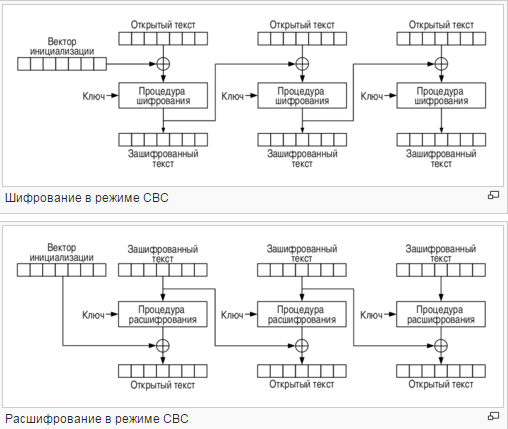
где i — номера блоков, IV — вектор инициализации ([синхропосылка](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BF%D0%BE%D1%81%D1%8B%D0%BB%D0%BA%D0%B0&action=edit&redlink=1" \o "Синхропосылка (страница отсутствует))), C_i и P_i — блоки зашифрованного и открытого текстов соответственно, а E_k — функция[блочного шифрования](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Расшифровка:

P_i=C_{i-1} \oplus D_k \left( C_{i} \right)

Для преодоления недостатков *ECB* используют способ, при котором одинаковые незашифрованные блоки преобразуются в различные зашифрованные. Для этого в качестве *входа алгоритма* используется результат применения операции *XOR* к текущему незашифрованному блоку и предыдущему зашифрованному блоку.

Для получения первого блока зашифрованного сообщения используется инициализационный вектор (IV), для которого выполняется операция *XOR* с первым блоком незашифрованного сообщения. При дешифровании для IV выполняется операция *XOR* с выходом дешифрирующего алгоритма для получения первого блока незашифрованного текста.

IV должен быть известен как отправителю, так и получателю. Для максимальной безопасности IV должен быть защищен так же, как ключ.



1. Описать и отобразить графически алгоритм CFB

**Режим обратной связи по шифротексту**, **режим гаммирования с обратной связью** (*CFB*) — один из вариантов использования симметричного [блочного шифра](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80), при котором для шифрования следующего блока [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) он [складывается по модулю 2](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2) с перешифрованным (блочным шифром) результатом шифрования предыдущего блока.

Шифрование может быть описано следующим образом:

C_0 = IV

C_i = E_k \left( C_{i-1} \right) \oplus P_i

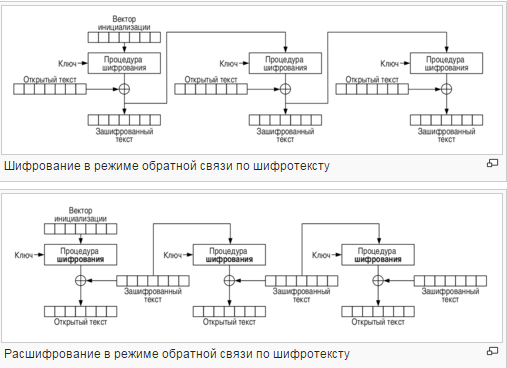
P_i = E_k \left( C_{i-1} \right) \oplus C_i

где i — номера блоков, IV — вектор инициализации ([синхропосылка](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BF%D0%BE%D1%81%D1%8B%D0%BB%D0%BA%D0%B0&action=edit&redlink=1" \o "Синхропосылка (страница отсутствует))), C_i и P_i — блоки зашифрованного и открытого текстов соответственно, а E_k — функция[блочного шифрования](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Блочный алгоритм предназначен для шифрования блоков определенной длины. Однако можно преобразовать блочный алгоритм в поточный *алгоритм шифрования*, используя последние два режима. Поточный *алгоритм шифрования* устраняет необходимость разбивать сообщение на целое число блоков достаточно большой длины, следовательно, он может работать в реальном времени. Таким образом, если передается поток символов, каждый символ может шифроваться и передаваться сразу, с использованием символьно ориентированного режима блочного алгоритма шифрования.

Рассмотрим шифрование. Входом функции шифрования является регистр сдвига, который первоначально устанавливается в инициализационный вектор IV. Для левых J битов *выхода алгоритма* выполняется операция *XOR* с первыми J битами незашифрованного текста Р1 для получения первого блока зашифрованного текста С1. Кроме того, содержимое регистра сдвигается влево на J битов, и С1 помещается в правые J битов этого регистра. Этот процесс продолжается до тех пор, пока не будет зашифровано все сообщение.

При дешифровании используется аналогичная схема, за исключением того, что для блока получаемого зашифрованного текста выполняется операция *XOR* с *выходом алгоритма* для получения незашифрованного блока.



1. Описать и отобразить графически алгоритм ОFB

Режим (OFB) обратной связи вывода превращает блочный шифр в синхронный шифр потока: он генерирует ключевые блоки, которые являются результатом сложения с блоками открытого текста, чтобы получить зашифрованный текст. Так же, как с другими шифрами потока, зеркальное отражение в зашифрованном тексте производит зеркально отраженный бит в открытом тексте в том же самом местоположении. Это свойство позволяет многим кодам с исправлением ошибок функционировать как обычно, даже когда исправление ошибок применено перед кодированием.

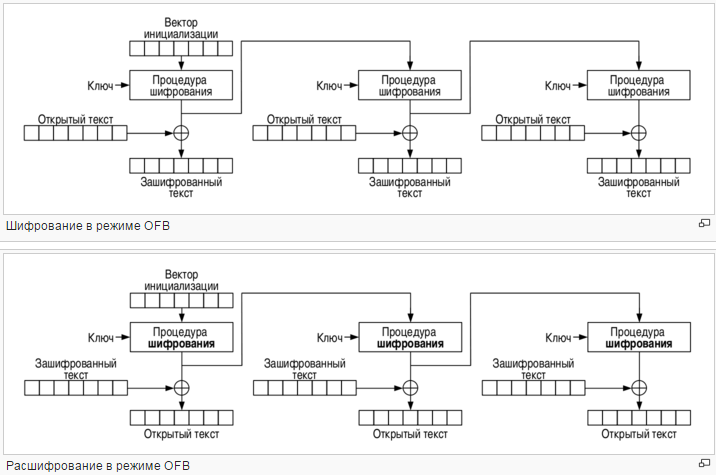
Из-за симметрии операции сложения, шифрование и расшифрование похожи:

C_i=P_i\oplus O_i

P_i=C_i\oplus O_i

O_i=E_k(O_{i-1})

O_0=IV



Каждая операция блочного шифра обратной связи вывода зависит от всех предыдущих и поэтому не может быть выполнена параллельно. Однако, из-за того, что открытый текст или зашифрованный текст используются только для конечного сложения, операции блочного шифра могут быть выполнены заранее, позволяя выполнить заключительное шифрование параллельно с открытым текстом.

Данный метод называется также «режим обратной связи по выходу».

OFB также предполагает некое усовершенствование касающееся метода генерации независимой последовательности блоков: для получения очередного блока предлагается шифровать не с O_i, а c O_i + IV(mod 2^{64}), где IV некоторый вектор инициализации.

1. Описать и отобразить графически схему формирования ключей для алгоритма RSA

RSA-ключи генерируются следующим образом:

1. Выбираются два различных [случайных простых числа](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) p и q заданного размера (например, 1024 [бита](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) каждое).
2. Вычисляется их произведение n=p\cdot q, которое называется *модулем*.
3. Вычисляется значение [функции Эйлера](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0) от числа n:

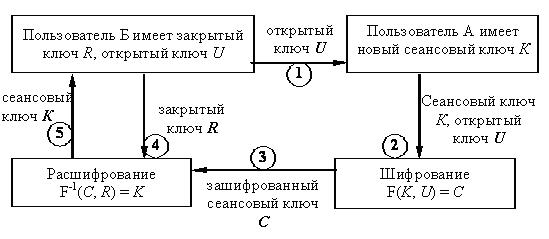
\varphi(n) = (p-1)\cdot (q-1).

1. Выбирается целое число e (1 < e < \varphi(n)), [взаимно простое](https://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%B0%D0%B8%D0%BC%D0%BD%D0%BE_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) со значением функции \varphi(n). Обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в [двоичной записи](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), например, простые [числа Ферма](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B0_%D0%A4%D0%B5%D1%80%D0%BC%D0%B0) 17, 257 или 65537.
   * Число e называется *открытой экспонентой* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *public exponent*)
   * Время, необходимое для шифрования с использованием [быстрого возведения в степень](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%B1%D1%8B%D1%81%D1%82%D1%80%D0%BE%D0%B3%D0%BE_%D0%B2%D0%BE%D0%B7%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D1%8C), пропорционально числу единичных бит в e.
   * Слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA.[[15]](https://ru.wikipedia.org/wiki/RSA#cite_note-Boneh-15)
2. Вычисляется число d, [мультипликативно](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F" \o "Мультипликативная функция) обратное к числу e по модулю \varphi(n), то есть число, удовлетворяющее [сравнению](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E):

d\cdot e \equiv 1 \pmod{\varphi(n)}.

* + Число d называется *секретной экспонентой*. Обычно, оно вычисляется при помощи [расширенного алгоритма Евклида](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%95%D0%B2%D0%BA%D0%BB%D0%B8%D0%B4%D0%B0).

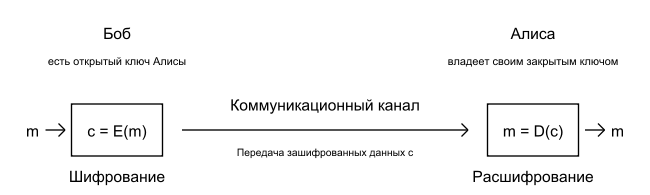
1. Пара \left\{ e, n \right\} публикуется в качестве *открытого ключа RSA* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *RSA public key*).
2. Пара \left\{ d, n \right\} играет роль *закрытого ключа RSA* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *RSA private key*) и держится в секрете.



1. Описать и отобразить графически алгоритм шифрования  и расшифровки сообщения по алгоритму RSA

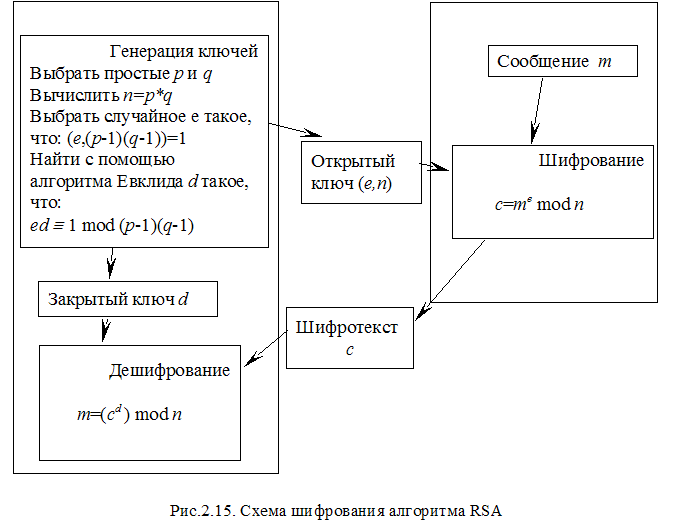
Предположим, Боб хочет послать Алисе сообщение m.

Сообщениями являются [целые числа](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) в интервале от 0 до n - 1, т.е m \in \mathbb{Z}_{n}\,.

[](https://commons.wikimedia.org/wiki/File:Public_key_encryption,_transmission_and_decryption_light-ru-rendered.svg?uselang=ru)

|  |  |
| --- | --- |
| **Алгоритм**:   * Взять *открытый ключ* (e,n) Алисы * Взять *открытый текст* m * Зашифровать сообщение с использованием открытого ключа Алисы:   c = E(m) = m^e \mod n ~~~~ (1) | **Алгоритм**:   * Принять зашифрованное сообщение c \, * Взять свой *закрытый ключ* (d,n) * Применить закрытый ключ для   расшифрования сообщения:  m = D(c) = c^d \mod n ~~~~(2) |

Данная схема на практике не используется по причине того, что она не является *практически надёжной*(semantically secured). Действительно, односторонняя функция E(m) является *детерминированной* — при одних и тех же значениях входных параметров (ключа и сообщения) выдаёт одинаковый результат —, а это значит, что не выполняется необходимое условие практической (семантической) надёжности шифра.



1. Описать и отобразить графически алгоритм работы функции MD2

Предполагается, что на вход подано сообщение, состоящее из ~b~ байт, хеш которого нам предстоит вычислить. Здесь ~b~ — произвольное неотрицательное целое число; оно может быть нулем или сколь угодно большим. Запишем сообщение побайтово, в виде:

m_0 m_1 \ldots m_{b-1}

Ниже приведены 5 шагов, используемые для вычисления хеша сообщения.

Шаг 1. Добавление недостающих бит.

Сообщение расширяется так, чтобы его длина в байтах по модулю 16 равнялась 0. Таким образом, в результате расширения длина сообщения становится кратной 16 байтам. Расширение производится всегда, даже если сообщение изначально имеет нужную длину.

Расширение производится следующим образом: i байт, равных i, добавляется к сообщению, так чтобы его длина в байтах стала равной 0 по модулю 16. В итоге, к сообщению добавляется как минимум 1 байт, и как максимум 15.

Шаг 2. Добавление контрольной суммы

16-байтная контрольная сумма сообщения добавляется к результату предыдущего шага.

Контрольная сумма вычисляется следующим образом: для каждого 16-байтного блока дополненного сообщения 16 раз выполняются следующие действия:

c=M[i*16+j],

C[j]=S[c \oplus L] \oplus C[j],

L=C[j]

, где i — номер 16-байтного блока данных;

j — номер текущего шага цикла;

M[x] — x-й байт сообщения, то есть M[i * 16 + j] — это j-й байт текущего блока данных;

с и L — временные переменные, L изначально содержит значение 0;

C[j] — j-й байт массива контрольной суммы; перед вычислением контрольной суммы массив содержит нулевые байты;

S[i] — i-й элемент 256-байтной матрицы из «случайно» переставленных цифр числа пи:

16-байтная контрольная сумма C[0 \ldots 15] добавляется к сообщению. Теперь сообщение можно записать в виде M[0 \ldots N_1-1], где N_1 = N + 16.

### Шаг 3. Инициализация MD-буфера

48-байтный буфер X используется для вычисления хеша. Он инициализируется нулями.

### Шаг 4. Обработка сообщения блоками по 16 байт.

На этом шаге используется та же 256-байтная перестановочная матрица S, как и на шаге 2.

Каждый 16-байтный i-й блок дополненного сообщения, включая блок контрольной суммы, накладывается на буфер X следующим образом:

1. Блок данных копируется в буфер следующим образом:

X[16 + j] = M[i * 16 + j], j = 0…15,

X[32 + j] = (X[16 + j] \oplus  X[j]), j = 0…15.

1. Выполняется обработка буфера, которая состоит из 18 раундов преобразований, в рамках каждого из которых выполняется обновление каждого байта буфера следующим образом:

X[k] = (X[k] \oplus S[t]), t = X[k],

где k = 0…47,

t — временная переменная, которая изначально имеет нулевое значение, а после выполнения каждого j-го раунда (j = 0…17) t модифицируется по правилу:

t = t + j \mod 256.

### Шаг 5. Формирование хеша.

Хеш вычисляется как результат X[0 \ldots 15]; в начале идет байт X[0], а конце X[15].

1. Описать и отобразить графически алгоритм работы функции MD4

Алгоритм MD4

Предполагается, что на вход подано сообщение, состоящее из ~b~ бит, хеш которого нам предстоит вычислить. Здесь ~b~ — произвольное неотрицательное целое число; оно может быть нулем, не обязано быть кратным восьми, и может быть сколь угодно большим. Запишем сообщение побитово, в виде:

m_0 m_1 \ldots m_{b-1}

Ниже приведены 5 шагов, используемые для вычисления хеша сообщения.

Шаг 1. Добавление недостающих битов.

Сообщение расширяется так, чтобы его длина в битах по модулю 512 равнялась 448. Таким образом, в результате расширения, сообщению недостает 64 бита до длины, кратной 512 битам. Расширение производится всегда, даже если сообщение изначально имеет нужную длину.

Шаг 2. Добавление длины сообщения.

64-битное представление ~b~ (длины сообщения перед добавлением набивочных битов) добавляется к результату предыдущего шага. В маловероятном случае, когда ~b~больше, чем 2^{64}, используются только 64 младших бита. Эти биты добавляются в виде двух 32-битных слов, и первым добавляется слово, содержащее младшие разряды.

На этом этапе (после добавления битов и длины сообщения) мы получаем сообщение длиной кратной 512 битам. Это эквивалентно тому, что это сообщение имеет длину, кратную 16-ти 32-битным словам. Каждое 32-битное слово содержит четыре 8-битных, но следуют они не подряд, а наоборот (например, из восьми 8-битных слов (a b c d e f g h) мы получаем два 32-битных слова (dcba hgfe)). Пусть M[0 \ldots N-1] означает массив слов получившегося сообщения (здесь N кратно 16).

Шаг 3. Инициализация MD-буфера.

Для вычисления хеша сообщения используется буфер, состоящий из 4 слов (32-битных регистров): ~(A,B,C,D). Эти регистры инициализируются следующими шестнадцатеричными числами (младшие байты сначала):

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Шаг 4. Обработка сообщения блоками по 16 слов.

Для начала определим три вспомогательные функции, каждая из которых получает на вход три 32-битных слова, и по ним вычисляет одно 32-битное слово.

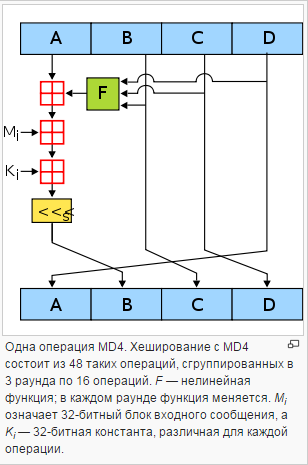
F(X,Y,Z) = XY \lor \neg X Z

G(X,Y,Z) = XY \lor XZ \lor YZ

H(X,Y,Z) = X \oplus Y \oplus Z

Шаг 5. Формирование хеша.

Результат (хеш-функция) получается как ABCD. То есть, мы выписываем 128 бит, начиная с младшего бита A, и заканчивая старшим битом D.



1. Описать и отобразить графически алгоритм работы функции MD5

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения может быть любой (в том числе нулевой). Запишем длину сообщения в *L*. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

Ниже приведены 5 шагов алгоритма:

Шаг 1. Выравнивание потока

Сначала дописывают единичный бит в конец потока (байт 80h), затем необходимое число нулевых бит. Входные данные выравниваются так, чтобы их новый размер L' был [сравним](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_%D0%BD%D0%B0%D1%82%D1%83%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) с 448 по модулю 512 (L' = 512 \times N + 448). Выравнивание происходит, даже если длина уже сравнима с 448.

Шаг 2. Добавление длины сообщения

В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта, затем старшие. Если длина превосходит 2^{64}-1, то дописывают только младшие биты (эквивалентно взятию по модулю 2^{64}). После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

Шаг 3. Инициализация буфера

Для вычислений инициализируются 4 переменных размером по 32 бита и задаются начальные значения шестнадцатеричными числами

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется инициализирующим вектором.

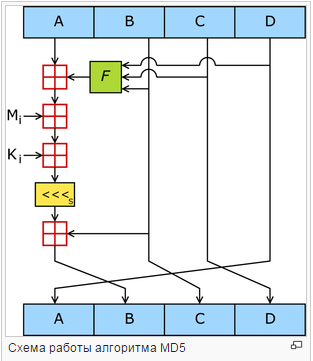
Шаг 4. Вычисление в цикле

Заносим в блок данных элемент *n* из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то переходим к следующему элементу массива (*n* + 1) и повторяем цикл.

Шаг 5. Результат вычислений

Результат вычислений находится в буфере ABCD, это и есть хеш. Если выводить побайтово, начиная с младшего байта A и закончив старшим байтом D, то мы получим MD5-хеш. 1, 0, 15, 34, 17, 18…



1. Описать и отобразить графически алгоритм работы функции SHA

Исходное сообщение разбивается на блоки по 512 бит в каждом. Последний блок дополняется до длины, кратной 512 бит. Сначала добавляется 1 (бит), а потом нули, чтобы длина блока стала равной (512 - 64 = 448) бит. В оставшиеся 64 бита записывается длина исходного сообщения в битах (в little-endian формате). Если последний блок имеет длину более 448, но менее 512 бит, то дополнение выполняется следующим образом: сначала добавляется 1 (бит), затем нули вплоть до конца 512-битного блока; после этого создается ещё один 512-битный блок, который заполняется вплоть до 448 бит нулями, после чего в оставшиеся 64 бита записывается длина исходного сообщения в битах (в little-endian формате). Дополнение последнего блока осуществляется всегда, даже если сообщение уже имеет нужную длину.

Инициализируются пять 32-битовых переменных.

A = a = 0x67452301

B = b = 0xEFCDAB89

C = c = 0x98BADCFE

D = d = 0x10325476

E = e = 0xC3D2E1F0

Определяются четыре нелинейные операции и четыре константы.

|  |  |  |
| --- | --- | --- |
| F_t(m, l, k) = (m \wedge l) \vee (\neg{m} \wedge k) | K_t = 0x5A827999 | 0≤t≤19 |
| F_t(m, l, k) = m \oplus l \oplus k | K_t = 0x6ED9EBA1 | 20≤t≤39 |
| F_t(m, l, k) = (m \wedge l) \vee (m \wedge k) \vee (l \wedge k) | K_t = 0x8F1BBCDC | 40≤t≤59 |
| F_t(m, l, k) = m \oplus l \oplus k | K_t = 0xCA62C1D6 | 60≤t≤79 |

Главный цикл

Главный цикл итеративно обрабатывает каждый 512-битный блок. Итерация состоит из четырех этапов по двадцать операций в каждом. Блок сообщения преобразуется из 16 32-битовых слов M_i в 80 32-битовых слов W_j по следующему правилу:

W_t  =  M_t  при 0≤t≤15  
W_t = (W_t-3 \oplus W_t-8 \oplus W_t-14 \oplus W_t-16) << 1 при 16≤t≤79

здесь << — это циклический сдвиг влево

для t от 0 до 79   
 temp = (a<<5) + F_t(b,c,d) + e + W_t + K_t    
 e = d   
 d = c   
 c = b<<30   
 b = a   
 a = temp

После этого a, b, c, d, e прибавляются к A, B, C , D , E соответственно. Начинается следующая итерация.

Итоговым значением будет объединение пяти 32-битовых слов в одно 160-битное хеш-значение.

