Данное пособие ориентировано на практическую поддержку специального курса «Криптографические методы защиты информации» и частично "Лабораторного практикума по информационно - компьютерным технологиям" для студентов (бакалавров и магистров), обучающихся по направлениям «Прикладная математика и информатика» и «Информационная безопасность». Студенты, завершившие обучение по данным направлениям, должны ориентироваться в методах криптографической защиты информации, обладать практическими навыками простейшей защиты информации в информационных системах. Лабораторный практикум по данной дисциплине предполагает реализацию курсового проекта, представляющего собой комплексное выполнение нескольких заданий по различным аспектам практической криптографии.

Представленные в данном пособии материалы, ориентированые на реализацию такого проекта, апробированы в течение нескольких лет в Институте вычислительной математики и информационных технологий Казанского (Приволжского) Федерального университета.

СОДЕРЖАНИЕ ПЕРВОЙ ЧАСТИ

СОДЕРЖАНИЕ ЧАСТЕЙ II-IV

I.1. Введение.
Содержание разделов курсового проекта по курсу «Криптографические методы защиты информации»

Без использования криптографии в настоящее время немыслимо решение задач по обеспечению безопасности информации, связанных с конфиденциальностью и целостностью, аутентификацией и невозможностью отказа от авторства. До 1990 года криптография обеспечивала закрытие лишь государственных линий связи. Однако в наше время использование криптографических методов получает широкое распространение благодаря развитию компьютерных сетей и электронного обмена данными в различных сферах человеческой деятельности: в промышленности, финансовом и банковском деле, торговле и т.п. Очевидно, что значение криптографических методов в указанных областях будет только возрастать.

Разработка и использование современных приложений криптографии невозможно без изучения теоретических основ криптографии. При этом теоретические курсы должны сопровождаться обязательными практическими занятиями по разработке программно-аппаратных средств защиты информации.

Цель данного пособия – обеспечить практическую поддержку специального курса «Криптографические методы защиты информации» и частично ″Лабораторного практикума по информационно-компьютерным технологиям″ для студентов (бакалавров и магистров), обучающихся по направлениям «Прикладная математика и информатика» и «Информационная безопасность». Студенты, завершившие обучение по данным направлениям, должны ориентироваться в методах криптографической защиты информации, обладать практическими навыками простейшей защиты информации в информационных системах. Лабораторный практикум по данной дисциплине предполагает реализацию курсового проекта, представляющего собой комплексное выполнение нескольких заданий по различным аспектам практической криптографии.

Общая структура реализуемого студентами курсового проекта представлена в следующей таблице:

Раздел	Наименование разделов курсового проекта «Криптографические методы защиты информации»
I.	**Реализация криптографических примитивов.** Реализация операций над n-битовыми блоками, длинной и модульной арифметики, вычислений в конечных полях, тестирование чисел на простоту, построение датчиков псевдослучайных чисел.
II.	**Скоростные блочные шифры.** Программная реализация процедур зашифрования и расшифрования на основе блочного шифра из предложенной коллекции современных блочных шифров.
III.	**Хеширование.** Построение сжатого образа сообщения.
IV.	**Электронная цифровая подпись.** Реализация алгоритмов вычисления электронной цифровой подписи сообщения и ее проверки на основе дискретных логарифмов или эллиптических кривых над конечными полями.
V.	**Режимы шифрования.** Зашифрование и расшифрование подписанного сообщения с использованием одного из режимов блочного или поточного шифрования.
VI.	**Построение псевдослучайных PIN–кодов.** Конструирование алгоритма генерации псевдослучайных десятичных PIN-кодов с использованием датчика случайных чисел криптографическим методом.

I.2. Реализация криптографических примитивов

Криптографический примитив – в широком смысле это операция, используемая в качестве элемента шифра (криптоалгоритма), в узком смысле это операция, определяющая требуемые свойства криптосистемы (например, стойкость и т.п.).

Современная криптография базируется на математическом аппарате теории чисел и алгебры. В данном разделе приводятся необходимые сведения и алгоритмы (а также некоторые задачи) из следующих областей:

1. Операции над n-битовыми блоками
2. Элементы теории чисел
3. Алгебраические структуры

2.1. Операции над n-битовыми блоками

Пусть $B_2 = \{0,1\}$ – алфавит двоичных цифр, называемых также *битами* [1].

Набор $B = b_{n-1}b_{n-2} \dots b_0$, составленный из n битов, т.е. $b_i \in B_2$, $0 \le i \le n-1$, будем называть *n-битовым блоком*. Число битов в блоке B называется *длиной* блока и обозначается $\lambda(B)$. Блок, составленный из m одинаковых битов c обозначается c^m. (Например, $0^8 = 00000000$, $1^8 = 11111111$.) Для обозначения четырехбитовых блоков используется шестнадцатеричная (16-ичная) система обозначений (нотация):

блок	16-ичное обозначение	10-ичное значение	блок	16-ичное обозначение	10-ичное значение
0000	0x0	0	1000	0x8	8
0001	0x1	1	1001	0x9	9
0010	0x2	2	1010	0xa	10
0011	0x3	3	1011	0xb	11
0100	0x4	4	1100	0xc	12
0101	0x5	5	1101	0xd	13
0110	0x6	6	1110	0xe	14
0111	0x7	7	1111	0xf	15

Такая система обозначений используется и для блоков, длина которых кратна 4. Например, блок

`1000 0000 1111 0110 1110 0111 1001 1010`

обозначается как `0x80f6e79a`. Для некоторых блоков традиционно используется следующая нотация: 4-битовый блок называется *полубайтом*, 8-битовый – *байтом* ($byte$), 16-битовый – *полусловом* ($word$), 32-битовый – *словом* ($longword$), а 64-битовый – *двойным словом* ($double\ word$). Впрочем, вместо термина «блок» будем использовать термин «слово», если из контекста ясно, какова длина блока.

Замечание. Формально множество всевозможных n-битовых блоков можно рассматривать как B_2^n – декартово произведение n экземпляров множества B_2:

$B_2^n = B_2 \times \dots \times B_2$ (n раз) $= \{b_{n-1}b_{n-2} \dots b_0 |\ b_i \in B_2,\ 0 \le i \le n-1\}$.

Всего имеется 2^n n-битовых блоков.

Для произвольного блока $B = b_{n-1}b_{n-2} \dots b_0 \in B_2^n$ положим

$$N(B) = b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \dots + b_0 2^0.$$

Число $N(B) \in \mathcal{N}_{2^n} = \{0,1,\dots,2^n-1\}$ называется *числовым значением* n-битового блока B. Бит b_{n-1} называется *старшим*, а b_0 – *младшими* битами в блоке B.

[1] Бит является минимальной двоичной единицей измерения энтропии и количества информации в ЭВМ, соответствующей одному двоичному разряду. Энтропия сообщения, выраженная в битах, определяется средним числом символов необходимых для записи этого сообщения. Определенное количество битов составляет размер других единиц – двоичных слов байта, килобайта, мегабайта и т.д.

Замечание. Существует извечный спор между «математиками» и «программистами» о том, как нумеровать при записи биты и байты. (Например, в системе IBM-360 биты в словах нумеровались, начиная с нуля: $b_0b_1 \dots b_{31}$, причем при представлении целых числе бит b_{31} рассматривался как младший.) Память современных компьютеров имеет байтовую организацию, а каждый байт имеет свой номер – адрес. С помощью одного байта можно представить неотрицательные целые числа от 0 до 255. Для чисел с большим значением необходимо несколько байтов. Чтобы расположить информацию в нескольких байтах, можно выбрать один из способов:

Big-endian – байт с наибольшей значащей частью (старший байт, в исходном тексте он находится слева) размещается в памяти по *наименьшему* адресу (в младшей адресной позиции). Такое представление называется по-русски *обратным порядком следования байтов*. В компьютерной литературе байт с большей значащей частью называют MSB ($Most\ Signiticant\ Byte$), байт с наименьшей значащей частью называется LSB ($Last\ Signiticant\ Byte$).

Little-endian – байт с наибольшей значащей частью (слева) размещается в памяти на *наибольшему* адресу. Такое представление называется по-русски *прямым порядком следования байтов.* [2]

Распространенные в России компьютеры с 32-процессорами используют архитектуру $little\text{-}endian$. Существуют, однако, и $big\text{-}endian$-компьютеры (IBM 370, $Motorolla$ 68000, Sun и многие $RISC$-процессоры). Система $Power\ PC$ допускает оба формата данных. В современном внеплатформенном языке $Java$ данные хранятся в формате $big\text{-}endian$, а ADA допускает возможность задания режима хранения многобайтовых данных.

***Пример* 1.** В записи $B = \mathtt{0xa1b2c3d4}$ – первый полубайт является старшим, и соответствующее числовое значение B равно

$N(B) = 10{\cdot}16^7 + 1{\cdot}16^6 + 11{\cdot}16^5 + 2{\cdot}16^4 + 12{\cdot}16^3 + 3{\cdot}16^2 + 13{\cdot}16^1 + 4{\cdot}160 = 2712847316$.

В $big\text{-}endian$-представлении байты блока B будут записаны в порядке `0xa1`, `0xb2`, `0xc3`, `0xd4`, а в $little\text{-}endian$ – в порядке `0xd4`, `0xc3`, `0xb2`, `0xa1`.

Отметим теперь некоторые операции над блоками.

2.1.1. Конкатенация блоков

Блок $C = a_{n-1}a_{n-2} \dots a_0b_{n-1}b_{n-2} \dots b_0$, получающийся в результате прописываия справа к блоку $A = a_{n-1}a_{n-2} \dots a_0$ битов блока $B = b_{n-1}b_{n-2} \dots b_0$, называется *конкатенацией* (сцеплением, соединением) блоков A и B и обозначается $C = A||B$ (или просто AB).

Например, если A =`01010101`=`0x55` и B =`11110011`=`0xf3`, то $A||B$ = `0101010111110011`=`0x55f3`. Очевидно, что $\lambda(A||B) = \lambda(A) + \lambda(B)$. Если $k|n$, то любой блок из B_2^n можно рассматривать как конкатенацию n/k блоков из B_2^n.

2.1.2. Побитовые логические операции над блоками

Пусть $\neg$ (not), & ($\otimes$ или and), $\vee$ ($\boldsymbol{or}$), $\oplus$(xor) – логические (булевы) операции, определенные на B_2 как

x	$\neg x$
0	1
1	0

x	y	$x\&y$	$x\vee y$	$x\oplus y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

(Названия операций: $\neg$ – *отрицание*, & – *конъюнкция*, или *логическое умножение*, $\vee$ – *дизъюнкция*, или *логическое сложение*, $\oplus$ – *сложение по модулю* 2.)

[2] В Лилипутии (см. книгу Джонатана Свифта «Приключения Гулливера» образовались две непримиримые партии по вопросу, с какого конца разбивать яйца – с тупого ($big\ side$) или острого ($little\ side$). Споры между сторонниками big- и $little\text{-}endian$ носят такой же характер.

Определим побитовые операции над блоками $A = a_{n-1}a_{n-2} \dots a_{0_{n-1}}$ и $B = b_{n-1}b_{n-2} \dots b_0$ одинаковой длины как

$\neg A = c_{n-1}c_{n-2}c_0$, где $c_i = \neg a_i$ (побитовое отрицание),

$A\&B = c_{n-1}c_{n-2}c_0$, где $c_i = a_i\&b_i$ (побитовое умножение),

$A\vee B = c_{n-1}c_{n-2}c_0$, где $c_i = a_i \vee b_i$ (побитовая дизъюнкция),

$A\oplus B = c_{n-1}c_{n-2}c_0$, где $c_i = a_i \oplus b_i$ (побитовое сложение по модулю 2),

$i = 0,1, \dots, n-1$. Такие операции (над байтами и словами длины 16, 32 и 64) обычно предусмотрены в современных языках программирования.

Отметим, что структура $(B_2^n, \oplus)$ является *группой*. Роль нуля играет нулевой блок 0^n; $a \oplus a = 0$, т.е. $-a \equiv a$. При заданных $a, b \in B_2^n$ уравнение $a \oplus x = b$ однозначно разрешимо относительно $x \in B_2^n$.

Множество B_2 с операциями & (·) и ⊕ образует конечное поле, которое обозначается как $GF(2)$ ($Galois\ field$ – поле Галуа), или $\mathbb{F}_2$. Множество $\mathbb{F}_2^n$ с операциями сложения (⊕) и умножения блоков на скаляр $c \in \mathbb{F}_2^n$ (по правилу $0 \cdot a = 0^n$, $1 \cdot a = a$, где $a \in \mathbb{F}_2^n$) образует векторное пространство над $\mathbb{F}_2^n$ размерности $dim\ \mathbb{F}_2^n = n$.[3] Множество $\mathbb{F}_2^n$ относительно операций сложения (⊕) и умножения блоков (&) образует линейную ассоциативную алгебру [4].

2.1.3. Сдвиги блоков

Операция shl_s сдвига влево на s битов, выполняемая над -битовым блоком $A = a_{n-1}a_{n-2} \dots a_0$, определяется как

$$shl_s(A) = \begin{cases} a_{n-(s-1)}a_{n-(s-2)} \dots a_0 0^s, \text{ если } s < n; \\ 0^n, \text{ если } s \geq n. \end{cases}$$

Сдвиг влево – это сдвиг в сторону старших битов, при этом старшие s битов выталкиваются за пределы разрядкой сетки и пропадают, а s младших битовых позиций заполняется нулями. Более того, эта операция трактуется как замена значения $N(A)$ на

$$N(shl_s(A)) = (N(A) \cdot 2^s)\ mod\ 2^n,$$

где $a\ mod\ b$ – остаток от деления a на b.

Операция сдвига вправо определяется как

$$shr_s(A) = \begin{cases} 0^s a_{n-1}a_{n-2} \dots a_{n-s}, \text{ если } s < n;; \\ 0^n, \text{ если } s \geq n. \end{cases}$$

Сдвиг вправо – это сдвиг битов в сторону младших битов (с выталкиванием s младших битов и заполнением s старших битовых позиций нулями). При этом

$$N(shr_s(A)) = N(A)\ div\ 2^s,$$

где $a\ div\ b$ – частное от деления a на b.

---

[3] Множество V называется *векторным пространством над полем* F, если для него выполнимы аксиомы:

V1. $(V, +)$ – абелева аддитивная группа;

V2. для любого вектора $\boldsymbol{v} \in V$ и любого элемента поля $c \in F$ определено произведение $c\boldsymbol{v}$ (элементы поля называют *скалярами*, а элементы V – *векторами*);

V3. выполняются дистрибутивные замены:

$c(\boldsymbol{u} + \boldsymbol{v}) = c\boldsymbol{u} + c\boldsymbol{v}, \forall c \in F, \forall \boldsymbol{u}, \boldsymbol{v} \in V,$

$(c + d)\ \boldsymbol{u} = c\boldsymbol{u} + d\boldsymbol{v}, \forall c, d \in F, \forall \boldsymbol{u} \in V;$

V4. выполняется ассоциативный закон:

$(cd)\ \boldsymbol{u} = c(d\boldsymbol{u}), \forall c, d \in F, \forall \boldsymbol{u} \in V.$

[4] Множество A называется *линейной ассоциативной алгеброй* над полем $\boldsymbol{F}$, если выполнены аксиомы:

A1. A – векторное пространство над $\boldsymbol{F}$;

A2. для любых $\boldsymbol{u}, \boldsymbol{v} \in A$ определено их произведение – элемент $\boldsymbol{uv} \in A$;

A3. выполняется ассоциативный закон:

$\boldsymbol{u}(\boldsymbol{vw}) = (\boldsymbol{uv})\boldsymbol{w}, \forall \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in A;$

A4. выполняются билинейные законы:

$\boldsymbol{u}(c\boldsymbol{v} + d\boldsymbol{w}) = c\boldsymbol{uv} + d\boldsymbol{uw}, c\boldsymbol{v} + d\boldsymbol{w})\boldsymbol{u} = c\boldsymbol{vu} + d\boldsymbol{wu}, \forall c, d \in \boldsymbol{F}, \forall \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in A,$

2.1.4. Циклические сдвиги блока A влево (rol_s) и вправо (ror_s) на s позиций определяются как

$$rol_s(A) = a_{n-s-1}a_{n-s=2} \dots a_0 a_{n-1}a_{n-2} \dots a_{n-s}, \text{ если } 0 \leq s < n;$$
$$ror_s(A) = a_{s-1}a_{s=2} \dots a_0 a_{n-1}a_{n-2} \dots a_s, \text{ если } 0 \leq s < n;$$
$$rol_t(A) = rol_s(A), ror_t(A) = ror_s(A), \text{если } s \equiv t \bmod n.$$

Для $0 \leq s < n$ справедливы соотношения:

$$rol_s(A) = shl_s(A) \vee shr_{n-s}(A),$$
$$ror_s(A) = shr_s(A) \vee shl_{n-s}(A),$$

где вместо $\vee$ можно использовать также $\oplus$.

2.2. Элементы теории чисел

2.2.1. Модулярная арифметика

Запись в модулярной арифметике $a \equiv b\ (mod\ n)$ читается "a сравнимо с по модулю n") означает, что числа a и b при делении на n дают один и тот же остаток, или $n|(a-b)$ (читается: "n делит $(a-b)$", или "$a-b$ делится на n без остатка"). Если $a \equiv b\ (\mathrm{mod}\ n)$, то b называют *вычетом* числа a по модулю n, а саму операцию нахождения вычета a по модулю n называют приведением числа a по модулю n.

Числа от 0 до $n-1$ образуют полную систему вычетов по модулю n. Это означает, что для любого целого a найдется $r \in \{0, 1, \dots, n-1\}$ такое, что $a \equiv r\ (mod\ n)$.

Модулярная арифметика над вычетами во многом аналогична обычной арифметике. Вычисляя значение некоторого выражения над целыми числами (с использованием операций сложения, вычитания и умножения) по модулю n, можно использовать следующие соотношения:

$$(a+b)\ mod\ n = [(a\ mod\ n) + (b\ mod\ n)]\ mod\ n,$$
$$(a-b)\ mod\ n = [(a\ mod\ n) - (b\ mod\ n)]\ mod\ n,$$
$$(a \cdot b)\ mod\ n = [(a\ mod\ n) \cdot (b\ mod\ n)]\ mod\ n,$$
$$(a \cdot (b+c))\ mod\ n = \{[(a \cdot b)\ mod\ n] + [(a \cdot b)\ mod\ n]\}\ mod\ n.$$

Другими словами, целые числа по модулю n с использованием операций сложения и умножения образуют коммутативное кольцо. Оно обозначается как $\mathbb{Z}_n$.

2.2.2. Арифметические операции

Операции сложения, вычитания и умножения $(A+B)\ mod\ m$, $(A-B)\ mod\ m$ и $(A \cdot B)\ mod\ m$ над целыми числами $A, B \in \mathbb{Z}$ по модулю $m \in \mathbb{N}$ будем для краткости обозначать как

$$A +_m B, A -_m B, A \cdot_m B.$$

Для n-битовых блоков $A, B \in B_2^n$ значение m часто выбирают равным 2^n. В этом случае будем использовать обозначения:

$$A \boxplus_n B \equiv A +_{2^n} B \equiv (A+B)\ mod\ 2^n,$$
$$A \boxminus_n B \equiv A -_{2^n} B \equiv (A-B)\ mod\ 2^n,$$
$$A \boxtimes_n B \equiv A \cdot_{2^n} B \equiv (A \cdot B)\ mod\ 2^n.$$

Операции $\boxplus_n$ и $\boxtimes_n$ реализуются как обычные арифметические операции над целыми числами (в двоичной системе счисления) с той лишь разницей, что в полученном результате оставляют n младших битов, а старшие отбрасывают. Отметим также, что

$$A \boxminus_n B = A \boxplus_n (not\ B) \boxplus_n C, N(\mathrm{C}) = 1.$$

Структура $(B_2^n, \boxplus_n, \boxtimes_n)$ является коммутативным кольцом (с делителями нуля при $n > 1$). Уравнение $A \boxplus_n X = B$ однозначно разрешимо относительно $X \in B_2^n \in$ для любых $A, B \in B_2^n$.

Уравнение $A \boxtimes_n X = B$ разрешимо относительно X тогда и только тогда, когда н.о.д. $(N(A), N(B)) = 1$, т.е. числа $N(A)$ и $N(B)$ взаимно просты.

2.2.3. Наибольший общий делитель

Пусть $a, b \in \mathbb{Z}$ – любые целые числа, не равные нулю одновременно. Наибольшее целое число, делящее одновременно числа a и b, называется их *наибольшим общим делителем* и обозначается НОД(a, b) (или $gcd(a, b)$ от $great\ common\ divisor$, или просто (a, b)). Отметим следующие свойства НОД:

1) $(a, 0) = |a|$;
2) $(a, b) = (b, a)$;
3) если $b \neq 0$, то $(a, b) = (r, b)$, где $r = a \bmod b$ – остаток от деления a на b.

Алгоритм Евклида вычисления НОД основан на использовании этих свойств:

$a := abs(a);\ b := abs(b);$
$\boldsymbol{while}\ (a > 0)\&(b > 0)\ do\ \{$
    $\boldsymbol{if}\ a > b\ \boldsymbol{then}\ a := a \bmod b\ \boldsymbol{else}\ b := b \bmod a$
$\};$
НОД$:= a + b.$

Алгоритм Евклида затрачивает на вычисление НОД(a, b) в худшем случае $O(log_2(|a| + |b|))$ времени (по числу арифметических операций). Анализ этого и других алгоритмов вычисления НОД см. у Д. Кнута (т. 2, §4.5.2, 4.5.3).

2.2.4. Быстрое возведение в степень

Пусть x – неотрицательное целое число, a и n – положительные целые числа. Значение $y = a^x \bmod n$ можно вычислить по схеме:

$y := 1;$
$\boldsymbol{for}\ i := 1\ \boldsymbol{to}\ x\ \boldsymbol{do}\ y := (y \cdot a) \bmod n.$

Но это медленный алгоритм, а при больших x заведомо неприемлемый. Существенно более быстрая схема основана на использовании следующего соотношения:

$$a^x \bmod n = (b^z \times c) \bmod n,$$

где $b = a^2 \bmod n$, $z = x\ div\ 2$, а $c = 1$ или a соответственно для четного и нечетного x. Данное соотношение подсказывает следующий алгоритм:

$y := 1;$
$\boldsymbol{while}\ x > 0\ \boldsymbol{do}\ \{$
    $\boldsymbol{if}\ x$ нечетно $\boldsymbol{then}\ y := (y \cdot a) \bmod n;$
    $a := (a \cdot a) \bmod n;$
    $x := x\ div\ 2$
$\}.$

Временнáя сложность (по числу арифметических операций) составляет для первого алгоритма $O(x)$, а для второго – $O(\log_2 x)$.

2.2.5. Мультипликативный обратный элемент по модулю n

Сравнение $ax \equiv 1\ (mod\ n)$ имеет решение x тогда и только тогда, когда a и n взаимно просты, т.е. $(a, n) = 1$. Если решение существует, то оно единственно в интервале $[0, n - 1]$ (как, впрочем, и в любом другом интервале $[b, b + n - 1]$). Значение x, удовлетворяющее данному сравнению, называется *мультипликативным обратным к* a *по модулю* n и обозначается $a^{-1}(mod\ n)$ (или просто a^{-1}, если из контекста ясно, каково n). Значение a^{-1} можно вычислить, основываясь на теореме Эйлера:

$$a^{\varphi(n)} \equiv 1 (mod\ n)\ \Rightarrow\ a^{-1} = a^{\varphi(n)-1} \bmod n,$$

где $\varphi(n)$ – функция Эйлера (количество чисел среди $1, 2, \ldots, n$, взаимно простых с n); в частности, если n – простое число, то

$$a^{-1} = a^{n-2} \bmod n.$$

Теорема Эйлера допускает следующее обобщение:

Теорема Кармайкла. Для любых взаимно простых чисел $a \in \mathbb{Z}, n \in \mathbb{N}$

$$a^{\lambda(n)} \equiv 1\ (mod\ n), \tag{1}$$

где $\lambda(n)$ – *функция Кармайкла*, определяемая следующим образом:

$\lambda(2) = 1$, $\lambda(4) = 2$; $\lambda(2^\alpha) = 2^{\alpha-2}$, если $\alpha \geq 3$;

$\lambda(p^\alpha) = \varphi(p^\alpha) = p^\alpha(p-1)$, если p – нечетное простое число;

$\lambda(n) =$ НОК $[\lambda(p_1^{\alpha_1}),\ldots, \lambda(p_k^{\alpha_k})]$, если $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ – каноническое разложение числа n, где НОК $[a_1,\ldots, a_k)]$ – наименьшее общее кратное чисел $a_1,\ldots, a_k$.

Таким образом, если a и n взаимно просты, то $a^{-1} = a^{\lambda(n)-1}\ mod\ n$.

Замечание. Для чисел n вида 2, 4, p^α или $2p^\alpha$, где p – простое число, $\alpha \in \mathbb{N}$, имеет место равенство $\lambda(n) = \varphi(n)$. Во всех остальных случаях $\lambda(n)$ – собственный делитель числа $\varphi(n)$ и для них соотношение (1) улучшает теорему Эйлера**.**

Общим недостатком вычисления $a^{-1}(mod\ n)$ на основе теорем Эйлера и Кармайкла является необходимость вычисления значений $\varphi(n)$ и $\lambda(n)$, что связано с разложением числа n на простые множители. Другой способ вычисления a^{-1}, свободный от этого недостатка, основан на использовании расширенного алгоритма Евклида. Следующий алгоритм возвращает значение $d = (a, n)$ и значение x, удовлетворяющее сравнению

$ax \equiv d\ (mod\ n)$ (поэтому, если $d = 1$, то $x = a^{-1}\ (mod\ n)$):

```
(d, m, y, x) := (a, n, 0,1);
r := m mod d;
while r > 0 do {
        q := m div d;
        z := (y + (n − ((q · x) mod n))) mod n;
        (m, d) := (d, r);
        (y, x) := (x, z);
        r := m mod d
}.
```

2.2.6. Тестирование чисел на простоту

Простейшим методом проверки простоты натурального числа n является метод пробных делений: для $d = 2, 3, 5, 7, \ldots$ проверяется выполнение условия $n\ mod\ d \neq 0$ или условия НОД$(n, d) = 1$. Если эти условия выполняются для каждого d, не превосходящего $\sqrt{n}$, то n – простое число, в противном случае n – составное число. Этот метод работает медленно. Поэтому для больших чисел он неприменим.

Для доказательства простоты числа n можно использовать следующее обращение Малой теоремы Ферма:

***Теорема Люка* (**1876**).** Натуральное число n является простым тогда и только тогда, когда существует число b такое, что $b^{n-1} \equiv 1 \pmod n$, но $b^{(n-1)/q} \not\equiv 1 \pmod n$ для любого простого делителя q числа $n-1$.

Из Малой теоремы Ферма следует, что если НОД$(b, n) = 1$ и $b^{n-1} \not\equiv 1 (\mathrm{mod}\ n)$, то n – заведомо составное число. Вместе с тем существуют составные числа, для которых

$$b^{n-1} \equiv 1 \pmod n.$$

Такие (составные) числа называют b-*псевдопростыми* по модулю n.

Теорема Чиполлы (1904**).** Существует бесконечно много b-псевдопростых чисел.

Недостаток теста проверки числа n на простоту на основе теоремы Люка заключается в необходимости разложения числа $n-1$ на простые множители. С другой стороны, этот тест полезен при конструировании простых чисел вида $bq + 1$, где q – известное простое число, а для четного b известно разложение на простые множители.

На практике обычно используют вероятностные тесты проверки простоты. Для этих тестов необходима последовательность равномерно распределенных случайных чисел из отрезка $[1, n]$. Для каждого случайного числа a проверяется выполнение некоторых условий. Если какое-либо условие не выполнено, то n – заведомо составное число. Если же все условия выполнены, то с некоторой вероятностью ошибки можно утверждать, что n – простое число. Вероятность ошибки тем ближе к 0, чем больше чисел a будет испытано. Наиболее широко используется тест Рабина-Миллера.

Определение. Пусть n – нечетное число, $n - 1 = 2^s t$, где t нечетно. Число n называется *сильным b-псевдопростым* (*сильным b-псп*) по модулю n, если либо

$$b^t \equiv 1 \text{ или } n-1 \ (mod\ n),$$

либо

$$(b^t)^k \equiv n-1 \ (mod\ n)$$

для некоторого $k = 2, 4, 8, \dots, 2^{s-1}$.

Экспериментально установлено, что число $n < 25 \times 10^9$ является простым тогда и только тогда, когда n – сильное b-псп для $b = 2, 3, 5$ и 7. Исключение составляет составное число $n = 3215031751 = 151 \times 751 \times 28351$. Если $n <$ 341 550 071 728 321 является сильным b-псп при $b = 2, 3, 5, 7, 11, 13$ и 17, то n – простое число. Для больших значений n тестирование на простоту опирается на следующую теорему:

Теорема Рабина. Если n – нечетное составное число и

$$S = \{b | 1 \le b \le n-1 \text{ и } n \text{ не является сильным -псп } \},$$

то

$$|S| \ge (3/4)(n-1).$$

Тест Рабина проводится так. Случайным образом выбираем k значений $b \in \{1,2,\dots,n-1\}$ и для каждого b проверяем, является ли n сильным b-псп. Если нет, то n – заведомо составное число; если да, то можно с вероятностью ошибки $\le ({}^1\!/_4)^k$ утверждать, что n – простое число. Известны и более точные оценки. Например, для 256-битового кандидата в простые числа вероятность ошибки при $k = 6$ испытаниях не превосходит $({}^1\!/_2)^{51}$. Отметим также, что во многих реализациях проверяется делимость n на все простые числа, меньшие некоторого числа. Например, проверка делимости на простые числа, меньшие 256, отсекает из числа кандидатов в простые числа 80% нечетных чисел; еще более надежна проверка делимости на простые числа, меньшие 2000.

2.3. Алгебраические структуры

2.3.1. Группы. Множество G с заданной на нем бинарной операцией $'' \cdot ''$ называется группой, если выполнены три условия (аксиомы):

G1. Операция $'' \cdot ''$ замкнута на G, т.е. $a \cdot b \in G$ для любых $a, b \in G$.

G2. Операция $'' \cdot ''$ ассоциативна, т.е. $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ для любых $a, b, c \in G$.

G3. Существует элемент $e \in G$ такой, что $e \cdot g = g \cdot e = g$ для любого $g \in G$.

G4. Для любого $g \in G$ существует $g' \in G$ такой, что $g \cdot g' = g' \cdot g = e$.

Обычно для группы используется обозначение $(G, \cdot)$. Элемент e называется *нейтральным элементом* группы G, а g' – *обратным элементом* к g. В группе нейтральный элемент и элемент, обратный к g, определены однозначно, а уравнения

$$a \cdot x = b, \qquad y \cdot a = b$$

однозначно разрешимы (первое относительно x, а второе – относительно y) при любых a, $b \in G$. Операция $''\cdot''$ называется *коммутативной*, если $a \cdot b = b \cdot a$ для любых $a, b \in G$. В этом случае группа называется *коммутативной,* или *абелевой*.

***Пример* 2.** 1) Множество целых чисел $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$ с операцией сложения по модулю n образует аддитивную группу порядка n; 2) множество целых чисел $\mathcal{G}_n = \{1, 2, \dots, n-1\}$, где n – простое число, с операцией умножения по модулю n а образует мультипливную группу порядка $n-1$.

2.3.2. Группы подстановок

Подстановкой непустого множества M называют любое *биективное* (т.е. взаимно однозначное) отображение множества M на себя. Множество всех подстановок на множестве M обозначается через $S(M)$.

Произведение $f \circ g$ отображений f и g определяется как $f \circ g\ (x) = f(g(x)$ для любого $x \in M$. Множество подстановок $S(M)$ образует относительно произведения $\circ$ группу. Если M – конечное множество мощности n, то $S(M)$ – группа порядка $n!$ Ее называют *симметрической группой степени* n.

Группа $S(M)$ коммутативна только при $n \leq 2$.

Замечание. Элементы конечного множества M можно занумеровать как $0,1,\dots,n-1$. Тогда вместо группы $S(M)$ можно рассматривать группу $S(\Omega)$, $\Omega = \{0,1,\dots,n-1\}$; последнюю группу обычно обозначают через S_n.

Любая подгруппа G группы S_n (G – подмножество в S_n, само являющееся группой) называется *группой подстановок степени* n.

Пусть $\sigma_1,\dots,\sigma_k$ – подстановки, заданные на элементах $b_1, b_2, \dots, b_k \in B_2^m$. Операция подстановки, применения к блоку $B = b_1||b_2||\dots||b_k$ с использованием подстановок $\sigma_1,\dots,\sigma_k$, заключается в замене блока B на блок $B' = b_1'\,||b_2'||\dots||b_k'$, где $b_i' = \sigma_i(b_i)$, $i = 1$, $2,\dots,k$.

***Пример* 3.** Рассмотрим подстановку на множестве полубайтов B_2^4, заданную таблицей:

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\sigma[x]$	5	a	f	c	b	6	9	d	1	7	e	2	3	4	8	0

Применение этой подстановки к полубайтам блока $B =$ `0x01234567` дает блок $B'=$ `0x5afcb69d`, повторное применение дает блок $B'' =$ `0x6y032974`.

Подстановка может быть задана таблицей, как в рассмотренном примере 3, либо аналитически. Например, отображение

$$x \longrightarrow ax + b\ (mod\ 256),$$

где $a -$ нечетное, является подстановкой на множестве B_2^8 байтов.

Если $\sigma -$ подстановка на элементах множества B_2^m, то обратная подстановка σ^{-1} (реализующая отображение, обратное к σ) может быть вычислена по схеме:

$$\boldsymbol{for}\ x \in B_2^m\ \boldsymbol{do}\ \{y := \sigma[x];\ \sigma^{-1}[y] := x\}.$$

***Пример* 4.** Подстановка σ^{-1}, обратная к подстановке σ, приведенной в примере 3, имеет следующий вид:

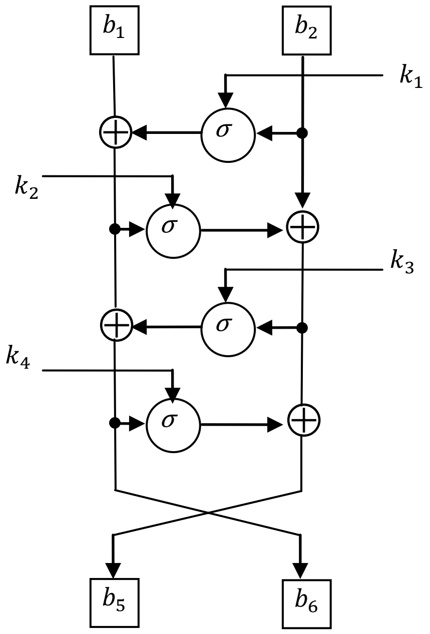
x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\sigma^{-1}[x]$	f	8	b	c	d	0	5	9	e	6	1	4	3	7	a	2

Подстановочные преобразования широко применяются при конструировании блочных шифров. Одно из требований, представляемых к используемой для замены подстановке σ, заключается в том, чтобы зависимость между x и $\sigma[x]$ была нелинейной и плохо апроксимировалась линейными функциями. В современных блочных шифрах чаще всего используют подстановки, заданные на множестве полубайтов (т.е. B_2^4) либо на множестве байтов (B_2^8). Это объясняется тем, что таблицы размера 16 (для случая B_2^4) и 256 (для случая B_2^8) поддаются экспериментальному исследованию на устойчивость по отношению к известным методам криптоанализа. Проектирование таблиц подстановок, имеющих больший размер, $-$ более сложная задача.

Отметим один способ конструирования подстановки Σ, заданной на множестве B_2^{16} двухбайтовых слов. Пусть $b_1 b_2 \in B_2^{16}$ – двухбайтовое слово; $\sigma_0,\dots,\sigma_{255}$ – любые подстановки на множестве B_2^8 байтов (построенная, например, псевдослучайным способом); $k_1, k_2, k_3,\ k_4 \in B_2^8$ – любые фиксированные байты (параметры, которые, наряду с σ_i, определяют конструируемую подстановку σ). Рассмотрим преобразование Σ: $B_2^{16} \longrightarrow B_2^{8}$, представленное на рис.1.

Это преобразование имеет обратное $\sum^{-1}$, представленное на рис.2, т.е. $\sum$ – подстановка на множестве двухбайтовых слов. Заметим, что подстановки $\sum$ и $\sum^{-1}$ вычисляются по одинаковой схеме, известной как 4-раундовая схема Фейстеля. Если первую подстановку обозначить как $\sum[k_1, k_2, k_3, k_4; \sigma_0, \sigma_1,..., \sigma_{255}]\ (b_1 b_2)$, то обратная подстановка задается как

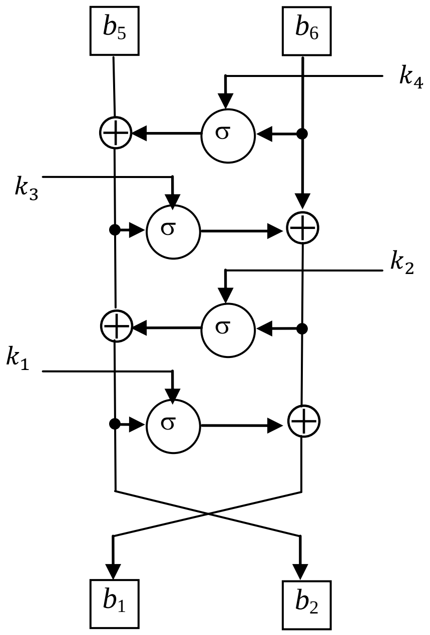
$$\sum^{-1}\ [k_1, k_2, k_3, k_4; \sigma_0, \sigma_1, ..., \sigma_{255}]\ (b_1 b_2)) = \sum[k_4, k_3, k_2, k_1; \sigma_0, ..., \sigma_{255}](b_1 b_2).$$



$Вход$: $b_1 b_2 \in B_2^{16}$.
$b_3 := \sigma_{k_1}[b_2] \oplus b_1$;
$b_4 := \sigma_{k_2}[b_3] \oplus b_2$;
$b_6 := \sigma_{k_3}[b_4] \oplus b_3$;
$b_5 := \sigma_{k_4}[b_6] \oplus b_4$;
$Выход$: $b_5 b_6 \in B_2^{16}$.

Рис. 1.

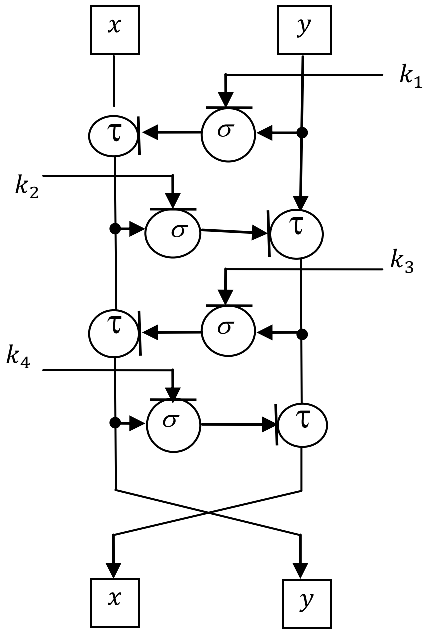
Другими словами, алгоритм вычисления $\sum^{-1}$ совпадет с алгоритмом вычисления $\sum$ с той лишь разницей, что при вычислении $\sum^{-1}$ ключевые параметры используются в обратном порядке (ключевые параметры k управляют выбором соответствующих подстановок σ_k).



$Вход$: $b_5 b_6 \in B_2^{16}$.
$b_4 := \sigma_{k_4}[b_6] \oplus b_5$;
$b_3 := \sigma_{k_3}[b_4] \oplus b_6$;
$b_2 := \sigma_{k_2}[b_3] \oplus b_4$;
$b_1 := \sigma_{k_1}[b_2] \oplus b_3$;
$Выход$: $b_1 b_2 \in B_2^{16}$.

Рис. 2.

Схемы, представленные на рис. 1, 2 можно обобщить путем замены операции $\oplus$ побитового сложения по модулю 2 на инволютивные подстановки $\tau_0, \tau_1, \ldots, \tau_{255}$. Соответствующее обобщение представлено на рис. 3. (Подстановка τ называется *инволютивной*, если $\tau^2 = e$, где e – тождественная подстановка, и, следовательно, $\tau^{-1} = \tau$. Такие преобразования представляют особый интерес для криптографии.)



Преобразование $\sum$:

Вход: $x||y \in B_2^{16}$.

$z := \sigma_{k_1}[y]$; $x := \tau_z[x]$;
$z := \sigma_{k_2}[x]$; $x := \tau_z[y]$;
$z := \sigma_{k_3}[y]$; $x := \tau_z[x]$;
$z := \sigma_{k_4}[x]$; $x := \tau_z[y]$;
$x \leftrightarrow y$.

Выход: $x||y \in B_2^{16}$.

Преобразование $\sum^{-1}$ реализуется аналогично, но ключевые параметры k_1, k_2, k_3, k_4 используются в обратном порядке.

Обозначение:

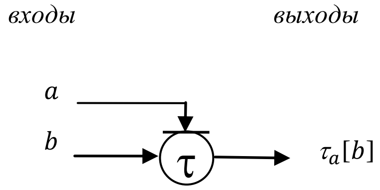


Рис. 3.

2.3.3. Перестановки битов

Операция π перестановки битов в блоке B_2^n, как и операция подстановки может быть задана таблицей или аналитически с помощью формул, указывающих в какую позицию перемещается тот или иной бит.

***Пример* 5.** Таблица

0	1	2	3	4	5	6	7
7	5	0	1	6	4	3	2

задает следующее преобразование: байт $b_7b_6b_5b_4b_3b_2b_1b_0$ преобразуется в байт $b_2b_3b_4b_6b_1b_0b_5b_7$, т.е. бит b_0 перемещается в позицию 2, бит b_1 – в позицию 3, бит b_2 – в позицию 7 и т.д.

2.3.4. Кольца

Множество R с заданными на нем операциями сложения $(+)$ и умножения $(\cdot)$ называется *кольцом*, если выполнены следующие условия (аксиомы):

$R1$. $(R, +)$ – абелева группа;

$R2$. Умножение замкнуто и ассоциативно на множестве R, т.е.

$a \cdot b \in R$ и $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ для любых $a, b, c \in R$;

$R3$. Выполняются дистрибутивные законы:

$(a + b) \cdot c = (a \cdot c) + (b \cdot c)$, $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ для любых $a, b, c \in R$.

Если операция $"\cdot"$ коммутативна, то кольцо R называется *коммутативным*.

***Пример* 6.**

Множество $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$, образующее полную систему вычетов целых чисел по модулю n с операциями сложения и умножения чисел по модулю n, является кольцом, причем коммутативным. Например, $\mathbb{Z}_4$ с операциями, заданными таблицами

$+_4$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	2	0	1	2

$\cdot_4$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

является коммутативным кольцом.

Нейтральный элемент кольца относительно операции $''+''$ называется *нулем* кольца и обозначается через 0. Для любого $x \in R$ имеем $x \cdot 0 = 0 \cdot x$.

Если существует нейтральный элемент относительно умножения, то этот элемент называется *единицей* и обозначается обычно через 1. Заметим, что единица существует не в каждом кольце. Например, в кольце четных целых чисел единица отсутствует. Если в кольце существует единица, то обратные элементы могут существовать не для каждого нулевого элемента. В рассмотренном кольце $\mathbb{Z}_4$ элемент 2 не имеет мультипликативного обратного (т.е. обратного относительно умножения).

2.3.5. Поля

Полем называется коммутативное кольцо с единицей, отличной от нуля, в котором каждый ненулевой элемент имеет мультипликативный обратный.

***Пример* 7.** Кольцо $\mathbb{Z}_n$ целых чисел по модулю n является полем тогда и только тогда, когда n – простое число. Другими примерами полей являются множества рациональных ($\mathbb{Q}$), действительных ($\mathbb{R}$) и комплексных ($\mathbb{C}$) чисел.

Для криптографии особый интерес представляют поля с конечным числом элементов. Число элементов конечного поля равно p^m, где p – простое число, $m \in \mathbb{N}$ – натуральное число. Поле из q элементов обозначается через $GF(q)$ или $\mathbb{F}_q$. (Обозначение GF образовано от $Galois\ field$, по имени Э. Галуа, первого исследователя конечных полей.)

Пусть $q = p^m$. Сформулируем основные свойства конечных полей:

Все элементы поля $\mathbb{F}_q$ являются корнями многочлена $x^q - x$.

Обозначим через $\mathbb{F}_q\,[x]$ множество многочленов с коэффициентами из поля $\mathbb{F}_q$. Множество $\mathbb{F}_q\,[x]$ относительно операций сложения и умножения многочленов является кольцом.

Пусть $deg\, f(x)$ обозначпает степень многочлена $f(x)$. Многочлен $f(x) \in \mathbb{F}_q\,[x]$ называется *неприводимым* над полем $\mathbb{F}_q$, если $f(x)$ не разлагается на нетривиальные множители, т.е. $f(x)$ не может быть представлен в виде $f(x) = g(x) \cdot h(x)$ ни при каких $g(x)$, $h(x) \in \mathbb{F}_q\,[x]$, степени которых удовлетворяют неравенствам $1 \le deg\, g(x)$, $deg\, h(x) < \deg f(x)$.

Любой неприводимый многочлен $f(x) \in \mathbb{F}_q\,[x]$ степени m является делителем многочлена $x^{q^m} - x$. Все его корни содержатся в поле $\mathbb{F}_{q^m}$, которое является *полем разложения* многочлена $f(x)$.

Любой делитель многочлена $x^{q^{m-1}} - 1$, неприводимый над полем $\mathbb{F}_q$, имеет степень, являющуюся делителем числа m.

В поле $\mathbb{F}_q$ существует элемент α такой, что всякий ненулевой элемент $\beta \in \mathbb{F}_q$ представим в виде $\beta = \alpha^i$ для некоторого $0 \le i \le q-2$. Другими словами, мультипликативная группа $\mathbb{F}_q^* = \mathbb{F}_q \backslash \{0\}$ поля $\mathbb{F}_q$, составленная из ненулевых элементов, является циклической, т.е. $\mathbb{F}_q^* = \{\alpha^0, \alpha^1, \dots, \alpha^{q-2}\}$. Элемент α называется *примитивным элементом* поля.

В прикладных задачах обычно используют задание поля либо в виде кольца классов вычетов целых чисел по простому модулю p (в этом случае $\mathbb{Z}_p = \mathbb{F}_p$), либо в виде фактор-кольца $\mathbb{F}_q\,[x]/f(x)$ – кольца многочленов $\mathbb{F}_q\,[x]$ по модулю неприводимого многочлена $f(x) \in \mathbb{F}_q\,[x]$.

В последнем случае конструкция поля $\mathbb{F}_{q^m}$, содержащего q^m элементов, описывается следующим образом. Предположим, что поле $\mathbb{F}_q$ построено, и пусть $f(x) \in \mathbb{F}_q\,[x]$ – неприводимый многочлен над $\mathbb{F}_q$ степени m. Тогда элементами поля $\mathbb{F}_{q^m}$ являются многочлены $g(x) \in \mathbb{F}_q\,[x]$, степень которых не превышает $m-1$. Сложение многочленов определяется как обычно: если $g(x) = \sum_i g_i x^i$, $h(x) = \sum_i h_i x^i$ то $g(x) + h(x) = \sum_i (g_i + h_i) x^i$ (где, конечно, сумма коэффициентов $g_i + h_i$ рассматривается в поле $\mathbb{F}_q$). Для умножения многочленов вводится понятие деления с остатком: разделить $g(x)$ на $f(x)$ значит представить многочлен $g(x)$ в виде:

$$g(x) = a(x) \cdot f(x) + r(x),\, deg\, r(x) < n,$$

где $r(x)$ – остаток от деления $g(x)$ на $f(x)$, определяемый однозначно.

По аналогии с целыми числами вводятся понятия вычета по модулю многочлена $f(x)$, сравнимость многочленов и операции сложения и умножения по модулю многочлена. Роль полной системы вычетов по модулю многочлена $f(x)$ выполняет множество всех возможных остатков от деления многочленов над полем $\mathbb{F}_q$ на $f(x)$. Другими словами, полную систему вычетов образуют многочлены

$$r(x) = r_0 + r_1 x + r_2 x^2 + \cdots + r_{m-1} x^{m-1};\; r_0,\, r_1, \ldots, r_{m-1} \in \mathbb{F}_q$$

(всего имеется q^m таких многочленов). Множество вычетов по модулю $f(x)$ с операциями сложения и умножения вычетов образуют коммутативное кольцо. Это кольцо является полем тогда и только тогда, когда $f(x)$ – неприводимый многочлен. Проще говоря, если перемножаются два многочлена (рассматриваемые как элементы поля $\mathbb{F}_{q^m}$), то в качестве результата необходимо перемножить их и взять остаток от деления произведения на модуль $f(x)$.

***Пример* 8.** 1) Простейшим конечным полем является поле $\mathbb{F}_2 = \{0,1\}$ из 2 элементов с операциями:

+	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

2) Построим поле из $4 = 2^2$ элементов. Многочлен $f(x) = x^2 + x + 1$ является неприводимым над полем $\mathbb{F}_2 = \{0,1\}$ из двух элементов. Элементами поля $\mathbb{F}_4$ являются $\{0, 1, x, x+1\}$, т.е. всевозможные многочлены с коэффициентами из $\mathbb{F}_2$ степени < 2. Таблицы сложения и умножения в поле $\mathbb{F}_4$ задаются следующим образом:

+	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

$\cdot$	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

2.3.6. Вычисления в конечном поле $\mathbb{F}_{256}$

Конечное поле $\mathbb{F}_{256}$, состоящее из 256 элементов, привлекательно для построения криптографических примитивов. В данном случае байты (8-битовые блоки) могут быть интерпретированы как элементы этого поля, а операции над элементами поля легко реализуемы.

Поле $\mathbb{F}_{256}$ можно рассматривать как фактор-кольцо $\mathbb{F}_2[x]/(f(x))$, где $f(x) \in \mathbb{F}_2[x]$ – неприводимый многочлен 8-ой степени.[5] На более простом языке это означает следующее. Элементы поля $\mathbb{F}_{256}$ представлены всевозможными многочленами

$$a(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

с коэффициентами из поля $\mathbb{F}_2 = \{0,1\}$, а многочлены в свою очередь, представлены 8-битовыми наборами (байтами) $a_7a_6a_5a_4a_3a_2a_1a_0$. Например, многочлен $x^6 + x^5 + x^3 + x + 1$ представлен двоичным набором 01101011 или байтом 0x6b (в 16-ичной записи). Операция сложения ($\oplus$) элементов поля $\mathbb{F}_{256}$ – это обычная операция сложения многочленов из $\mathbb{F}_2[x]$. Поскольку многочлены представлены байтами, то в данном случае сложение – это побитовое сложение байтов по модулю 2 (операция $\oplus$, или xor).

Операция умножения ($\odot$) в поле $\mathbb{F}_{256}$ реализуется сложнее, а именно: $a(x)\odot b(x) = c(x)$, где $c(x) = a(x)b(x)\ mod\ f(x)$ – остаток от деления многочлена $a(x)b(x)$ на $f(x)$.

***Пример* 9.** Пусть $f(x) = x^8+x^4+x^3+x^2+1$ – многочлен, на основе которого определяется конкретная реализация поля $\mathbb{F}_{256} \cong \mathbb{F}_2[x]/(f(x))$ (многочлен $f(x)$ является неприводимым в $\mathbb{F}_2[x]$, т.е. не разлагается на множители), и пусть

$$a(x) = x^7+x^5+x^1+1 \text{ и } b(x) = x^6+x^5+x^4+x^3+x^1+1$$

– многочлены, представляющие элементы данного поля. Тогда

$$a(x) \oplus b(x) = x^7+x^6+x^4+x^3,$$
$$a(x) \odot b(x) = x^7 + x^3 + x^1 + 1$$

(поскольку $a(x)b(x) = x^{13} + x^{12} + x^6 + x^5 + x^3 + x^2 + 1 = (x^5 + x^2 + x + 1)f(x) + x^5 + x^2 + x^1 + 1$), или, на "языке байтов", $a(x) = 0\text{x}a3$, $b(x) = 0\text{x}7b$;

$$a(x) \oplus b(x) = 0\text{x}a3 \oplus 0\text{x}7\text{b} = 0\text{x}d8,$$
$$a(x) \odot b(x) = 0\text{x}27.$$

Вычисление многочлена $c(x) = a(x)b(x)\ mod\ f(x)$ сводится к вычислению

$$a(x) \cdot x^m\ mod\ f(x) = (((a(x)\cdot x)\ mod\ f(x)) \cdot x^{m-1})\ mod\ f(x),$$

причем

$$(a(x)\cdot x)\ mod\ f(x) = \begin{cases} a(x)\cdot x, & \text{если } a_7 = 0, \\ a(x)\cdot x \ominus f(x), & \text{если } a_7 = 1. \end{cases}$$

где $a(x)\cdot x \ominus f(x) \equiv a(x)\cdot x \oplus f(x)$, поскольку в поле характеристики 2 имеем: $a \oplus a = 0$, и, следовательно, $a = -a$, $\forall a \in \mathbb{F}_{256}$.

Пусть $g(x)$ – многочлен 7-й степени такой, что $f(x) = x^8 + g(x)$; будем считать, что многочлен $g(x)$ представлен байтом g. Тогда вычисление $c = ab$ – произведения элементов $a, b \in \mathbb{F}_{256}[x]$ – можно выполнить по схеме (где все элементы – байты):

```
c := 0x00;
mask := 0x01;
for i := 1 to 7 do {
    if (b & mask) ≠ 0x00 then c := c⊕a;
    a := shl_1(a)
    if (a & 0x80) ≠ $00 then a := shl_1(a)⊕g;
    mask := shl_1(mask)
}.
```

Здесь $shl_1(x)$ – сдвиг битов байта x на одну позицию влево.

Значение $c = a^m (a, c \in \mathbb{F}_{256}, m \in N)$ вычисляется по быстрой схеме:

---

[5] Многочлен $f(x)$ необходимо зафиксировать. Выбор другого многочлена приведет в некоторому полю $\mathbb{F}'_{256}$ с другим представлением элементов. С алгебраической точки зрения поля $\mathbb{F}_{256}$ и $\mathbb{F}'_{256}$ изоморфны, т.е. отличаются только обозначениями элементов. Тем не менее поля $\mathbb{F}_{256}$ и $\mathbb{F}'_{256}$ следует считать различными, поскольку элементы этих полей могут участвовать в суррогатных вычислениях (т.е. в рамках разных вычислительных систем). Например, элементы поля $\mathbb{F}_{256}$, представленные байтами, могут интерпретироваться как элементы кольца $\mathbb{Z}_{256}$ целых чисел по модулю 256, и как элементы мультипликативной группы целых чисел по модулю 257 и т.п. Итоговый результат таких смешанных вычислений зависит от того, как согласованы представления элементов.

```
c := 0x01;
while m > 0 do {
    if m нечетно then c := c⊙a;
    a := a⊙a;
    m := m div 2
}.
```

Отметим, что $0^0 = 1$ и $0^m = 0$ для $m \geq 1$; если $a \neq 0$, то $a^{-n} = a^{255-n}$ ввиду $a^{255} = 1$.

Мультипликативный порядок $ord(a)$ ненулевого элемента $a \in \mathbb{F}_{256}$ определяется как наименьшее $m \in \mathbb{N}$, для которого $a^m = 1$. Значение $ord(a)$ можно вычислить по схеме:

```
ord := 255; p1 := 3; p2 := 5; p3 := 17;
for i := 1 to 3 do {
    m := ord div p_i;
    if a^m = 1 then ord := m
}.
```

В поле $\mathbb{F}_{256}$ имеется 128 элементов порядка 255; такие элементы называются *примитивными*. Пусть ω – один из них. Поиск ω можно осуществить по схеме:

```
ω := 0x02;
while ord(ω) < 255 do ω := ω + 1,
```

где символ + обозначает обычное арифметическое сложение байтов. Любой другой примитивный элемент ω′ может быть вычислен как

$$\omega' := \omega^m,$$

где m – число, взаимно простое с 255, т.е. НОД$(m, 255) = 1$. Тот факт, что $ord(\omega) = 255$, означает, что мультипликативная группа $\mathbb{F}_{256}^*$, состоящая из ненулевых элементов поля $\mathbb{F}_{256}$, является циклической, т.е.

$$\mathbb{F}_{256}^* = \mathbb{F}_{256} \setminus \{0\} = \{\omega^0, \omega^1, \dots, \omega^{254}\}.$$

Целое число j $(0 \leq j \leq 254)$ такое, что $a = \omega^j$ называется *дискретным логарифмом элемента* $a \in \mathbb{F}_{256}^*$ *по основанию* ω и обозначается $ind_\omega a$. Операции умножения и возведения в степень можно ускорить, если построить две вспомогательные таблицы $TD[0..254]$ и $TL[1..255]$, определяемые как

$$TD[j] = \omega^j, TL[\omega^j] = j, j = 0,1, \dots, 254.$$

Таблицы заполняются следующим образом:

```
a := 1;
for j := 0 to 254 do {
    TL[a] := j;
    TD[j] := a;
    a := a⊙ω
}.
```

Для любых $a, b \in \mathbb{F}_{256}^*$ имеем:

$$a \odot b = TD[(TL[a] + TL[b]) mod\ 255];$$
$$a^m = TD[(m \cdot TL[a]) mod\ 255], m \in Z,$$

в частности,

$$a^{-1} = \begin{cases} 1, \text{если } a = 1, \\ TD\big[255 - TL[a]\big], \text{если } a \neq 1. \end{cases}$$

Приведем две функции на языке Pascal, реализующие умножения и возведения в степень в конечном поле $\mathbb{F}_{256} \cong \mathbb{F}_2[x]/(f(x))$.

Элементы поля представлены многочленами $a(x) = a_7x^7 + a_6x^6 + \dots + a_1x^1 + a_0$, а в памяти ЭВМ – байтами $a = (a_7a_6 \dots a_1a_0)_2$. Сложение элементов поля – побитовое сложение байтов по модулю 2 (xor). Умножение элементов поля $a(x)$ и $b(x) \in \mathbb{F}_{256}[x]$ осуществляется так: $a(x)$ и $b(x)$ перемножаются, затем результат приводится по модулю неприводимого многочлена $f(x) = x^8 + f_7x^7 + f_6x^6 + \cdots + f_1x^1 + f_0 \in \mathbb{F}_{256}[x]$, т.е. находится остаток от деления $a(x) \cdot b(x)$ на $f(x)$. Многочлен $f(x)$ представлен байтом $f = (f_7f_6 \dots f_1f_0)_2$.

```
Function MulGF256(a, b, f: byte): byte;
// Для a(x), b(x) возвращает c(x) = a(x) · b(x) mod f(x).
var t, mask: byte;  i: integer;
begin
        t: = 0; mask: = 1;
        for i :=  0 to 7 do
        begin
                if(b and mask) <> 0 then t: = t xor a;
                if (a and 128) = 0 then a: = a shl 1 else a: = (a shl 1) xor f;
                mask := mask shl 1;
        end;
        MulGF256: = t;
end;
```

```
Function PowerGF256 (a, b, f: byte): byte;
// Возведение в степень: возвращает a(x)^b mod f(x).
var c: byte;
begin
        c: =  1;
        while b > 0 do begin
                if odd(b) then c: = MulGF256(c, a, f);
                a: = MulGF256(a, a, f); b: =  b shr 1;
        end;
        PowerGF256: = c;
end;
```