Реализация умножения многочленов в конечном поле

Вопрос задан 4 года 2 месяца назад Последняя активность 3 года 10 месяцев назад Просмотрен 1k раз

1

На зарубежном stackoverflow нашел пример реализации умножения многочленов в конечном поле Галуа. Помогите понять принцип работы этого алгоритма. Источник.

Код функции:

```
/* Multiply two numbers in the GF(2^8) finite field defined
 * by the polynomial x^8 + x^4 + x^3 + x + 1 */
uint8_t gmul(uint8_t a, uint8_t b) {
    uint8_t p = 0;
    uint8_t counter;
    uint8_t hi_bit_set;
    for (counter = 0; counter < 8; counter++) {
            if (b & 1)
                    p ^= a;
            hi_bit_set = (a & 0x80);
            a <<= 1;
            if (hi_bit_set)
                    a ^= 0x1b; /* x^8 + x^4 + x^3 + x + 1 */
            b >>= 1;
    }
    return p;
}
```

Математическое описание простое. Кольцо многочленов факторизуется по идеалу неприводимого многочлена f, в результате получается множество классов эквивалентности, которые состоят из многочленов, дающих одинаковый остаток от деления на f.

Математическое описание умножения многочленов поля Галуа: перемножаем два многочлена и берем остаток от деления результата на f.

Мы будем использовать только многочлены с коэффициентами в двоичном поле, то есть коэффициенты всех возможных многочленов - нули и единицы. Это удобно для реализации не только на языках, но и в виде электронных схем.

На низкоуровневых языках мы работаем с многочленами как с числами: каждому многочлену соответствует вектор его коэффициентов, который можно рассмотреть как разложение некоторого числа по основанию 2. Например, многочлену x^4 + x^2 + x + 1 можно поставить в соответствие его вектор коэффициентов 00010111, который является числом 23 в двоичной системе.

Помогите понять, как то, что я сейчас описал, реализуется этой функцией. Еще очень интересно, для чего служит переменная hi_bit_set, в которую при помощи константы 0x80 вырезается старший бит 8-битного числа.

с математика криптография

изменён 8 дек '15 в 20:57
Yuri Negometyanov
4,956 ● 12 ● 39

задан 15 авг '15 в 21:32
typemoon
1,104 ● 1 ● 11 ● 27

1 ответ

5

Во-первых, вы должны сами понимать, почему достаточно рассматривать многочлены степени не выше 7. Поэтому каждому многочлену можно поставить в соответствие 8-битное число. Отсюда тип данных `uint8_t`.

Рассмотрим основной цикл. В нём `counter` нужен лишь для того, чтобы выполнить цикл 8 раз.

```
for (counter = 0; counter < 8; counter++) {
```

Затем, что происходит с `b`? У него анализируется на каждой из итераций следующий бит. Вначале это младший бит, но на каждом шаге `b` сдвигается вправо на 1 бит, так что `b & 1` на `i`-ом есть `i`-ый бит первоначального числа `b`. (То есть, по сути, коэффициент при `i`-ой степени во втором многочлене.)

```
    if (b & 1)
```

В `p` накапливается текущая сумма. Если текущий коэффициент равен 1, к `p` прибавляется `a`, сдвинутое (см. ниже) влево на `i` единиц. XOR (`^`) равносилен сложению по модулю 2 (очевидно).

```
        p ^= a;
```

Запоминаем текущий верхний бит в `a` (то есть, старший коэффициент):

```
    hi_bit_set = (a & 0x80);
```

и сдвигаем `a` на один бит влево.

```
    a <<= 1;
```

Таким образом, в `a` получается на каждом шаге сдвинутое на 1 начальное значение (аналог `i`-ой строчке при умножении в столбик). При этом мы теряем старший бит, если он был равен 1 (т. к. гаш тип данных восьмибитный, и включает коэффициенты от 0 до 7 степени. Сейчас мы сделаем компенсацию для этого.

Если старший бит был установлен (то есть, коэффициент при восьмой степени не 0), «вспоминаем» о факторизации и вычитаем (или что то же самое по модулю 2, прибавляем) к `a` многочлен, по которому происходит факторизация. При этом коэффициент при 8-ой степени становится равным нулю, так что мы снова «влезаем» в наш восьмибитный тип данных.

```
    if (hi_bit_set)
        a ^= 0x1b; /* x^8 + x^4 + x^3 + x + 1 */
```

Конец итерации. Переходим к следующему коэффициенту.

```
    b >>= 1;
}
```

Мы видим, что у нас просто алгоритм умножения в столбик.

ответ дан 16 авг '15 в 11:31

VladD

185k ● 18 ● 233 ● 440

Спасибо за такой подробный ответ. Вчера думал над этим кодом и решил, что переменная hi_bit_set служит для определения того, надо или не надо приводить результат умножения по модулю. Если старший бит равен 1, это значит, что мы получили многочлен 8-й степени. Так как модуль у нас тоже многочлен 8-й степени, надо взять остаток от деления. Сделать это проще всего как при устном вычислении остатков от деления маленьких целых чисел: например, 4%3 = 4-3 = 1. Здесь мы тоже вычитаем модуль, что равносильно сложению с модулем по модулю 2. – typemoon 16 авг '15 в 11:43

А зачем мы анализируем в b каждый следующий бит? Зачем накапливаем сумму в p? Какие математические действия над многочленами соответствуют этим операциям? – typemoon 16 авг '15 в 11:43

@typemoon: ну, здесь если после сдвига получился многочлен 8-ой степени, к нему прибавляется модуль (что не меняет значение), и в результате получается многочлен не выше 7 степени. Это легче, чем брать остаток от деления. (По модулю 2 сложение и вычитание — одно и то же.) – VladD 16 авг '15 в 11:58

Анализ бита в `b` — это мы выясняем, получим мы многочлен 8-ой степени или нет. В `p` накапливается промежуточная сумма, как при умножении в столбик. – VladD 16 авг '15 в 11:59

Старший бит = коэффициент при 7-ой степени, а сумме соответствует сумма. Прогоните алгоритм вручную на каких-нибудь данных, станет понятнее. – VladD 16 авг '15 в 12:00