Московский Государственный Технический Университет им. Н. Э. Баумана.

Факультет информатики и систем управления.
Кафедра информационной безопасности.

Курсовая работа по дисциплине «Криптографические методы и средства защиты информации».

Программа для проведения вычислений в конечном поле «Euclid».

Выполнил: студент группы ИУ8-111
Солдатов Сергей.
Преподаватель: ***Жуков А. Е.***

Москва, 2001.

Оглавление.

1. Введение.

Целью данной работы является создание программного средства для проведения вычислений в конечном поле, являющимся расширением простого поля F_p с помощью неприводимого многочлена $g(x)$. Элементы поля рассматриваются как многочлены, представляющие собой остатки от деления на $g(x)$, или, что эквивалентно, как элементы фактор-кольца $F_p[x]/g(x)$.

Кроме проведения вычислений над многочленами, как элементами конечного поля $F_p[x]/g(x)$, при проектировании была предусмотрена возможность осуществления операций над многочленами, как многочленами над простым полем, т.е. как элементами кольца многочленов $F_p[x]$.

В данной работе сначала будут рассмотрены некоторые математические основы реализованных в работе алгоритмов, затем будет сказано несколько слов об их реализации на языке высокого уровня с использованием объектно-ориентированного подхода к программированию.

2. Теоретические сведения.

В данном разделе внимание будет заострено лишь на алгоритмах, которые так или иначе требуют какого-либо математического обоснования или выходят за рамки обычного перебора. Реализация простых, на взгляд автора, алгоритмов здесь не изложена.

2.1. Выполнение деления.

Поскольку реализация умножения и сложения тривиальны и не должны вызывать никаких вопросов, приступим сразу к изложению вопроса деления.

Деление в кольце многочленов над полем не вызывает затруднений, поскольку представляет собой обычное деление «столбиком», программная реализация которого не требует каких-либо математических знаний, кроме, конечно, представления о том, как вообще можно делить многочлен на многочлен.

Что касается деления в поле, то здесь реализация уже совершенно иная, так как связана с поиском обратного элемента в поле для делителя. Как известно, найти обратный элемент в поле можно по алгоритму Евклида[11)], реализуя, так называемое, евклидово деление. Из расширенного алгоритма Евклида, получаем:

---

[1]) Описанный здесь алгоритм применяется и при выполнении деления в простом поле. Разница лишь в строении структуры *PV*

$$nod(f(x), g(x)) = A(x)f(x) + B(x)g(x) \quad (1)$$

где *nod(f(x), g(x))* – НОД указанных многочленов.

Если *nod(f(x),g(x))=1*, получаем:

$$1 = A(x)f(x) + B(x)g(x) \Leftrightarrow A(x)f(x) = 1 mod g(x) \Leftrightarrow A(x) = f^{-1}(x) \quad (2)$$

Для реализации алгоритма Евклида определим следующую структуру, представляющую собой вектор, размера 2, элементами которого являются многочлены. Назовем ее *PV*[2]*:*

PV = {V1(x), V2(x)}, где *V1(x)* и *V2(x)* – полиномы. На множестве таких *PV* определим операции вычитания и умножения на полином следующим образом:

$$PV_2 = PV \cdot a(x) = \{V1(x) \cdot a(x), V2(x) \cdot a(x)\} \quad (3)$$

$$PV_3 = PV_1 - PV_2 = \{V1_1(x) - V1_2(x), V2_1(X) - V2_2(X)\} \quad (4)$$

С использованием структуры *PV* очень просто получить представление НОД в виде (1). Допустим надо найти представление (1) для многочленов *f(x)* и *g(x)*. Для этого необходимо сделать следующее:

1) Начальные данные:
   a) *N1:=f(x); N2:=g(x);*[3]*(deg f(x) deg g(x),* в противном случае – поменять *f(x)* и *g(x)* местами.
   b) *n1:={one, null}; n2:={null, one},* - где *n1, n2* – структуры типа *PV, null* – полином, состоящий из одного свободного члена, равного *0*, а one - полином, состоящий из одного свободного члена, равного *1*.
2) Разделим *N1(x)* на *N2(x)*, получим *Q(x)* и *R(x)* – соответственно частное и остаток.
3) *N1(x):=N2'(x); N2(x):=R(x)*;
4) *n2:=n1' – Q(x)·n2'*; *n1:=n2'*, где значения со штрихом *(')* означают значения, пришедшие с предыдущего шага алгоритма.
5) Шаги 2), 3) и 4) повторять до тех пор, пока *N2* не равно многочлену *null*.

После выхода из цикла вычислений *nod(f(x), g(x))* будет находиться в *N1*, а коэффициенты разложения (1) *A(x)* и *B(x)* в *n1* следующим образом:

$$\text{Если } n1 = \{V1, V2\}, \Rightarrow \{ \quad (5)$$

Вариант (*) выбирается в случае, если в пункте 1а алгоритма *f(x)* и *g(x)* менялись местами, в противном случае выбирается вариант (**).

---

[2]) От слов Polynomial и Vector

[3]) знак «:=» следует интерпретировать как «переменной слева присвоить значение переменной справа»

Доказательство этого алгоритма заключается в элементарной его проверке. Поскольку НОД, вычисленный в результате алгоритма Евклида, есть последний ненулевой остаток в евклидовом делении, то алгоритм представляет собой просто выражение каждого следующего остатка в алгоритме Евклида через предыдущие, а структура *PV* введена просто для удобства накопления выражения для НОД.

2.2. Тест на неприводимость.

В программе тест на неприводимость реализуется двумя различными способами, зависящими от степени тестируемого многочлена (см. далее.). Рассмотрим эти тесты.

2.2.1. Тест Берлекемпа.

Тест Берлекемпа является простым и универсальным, корректно работающим независимо от модуля q и степени тестируемого многочлена. Тест Берлекемпа сформулируем как теорему.

Теорема 1.

Пусть q – простое натуральное число. $Q(x) \in F_q[x]$, имеет степень n. *Q(x)* – неприводим тогда и только тогда, когда для любого простого делителя p числа n выполнено:

$Q(x)|(x^{q^n} - x)$ и НОД$(x^{q^{\frac{n}{p}}} - x, Q(x)) = 1.$

(6)

Доказательство.

Доказательство опирается на тот факт, что для неприводимого многочлена $Q(x) \in F_q[x]$ выполнено:

$$Q(x)|(x^{p^n} - x) \Leftrightarrow deg(Q)|n. \quad (7)$$

Допустим, что $Q(x)$ неприводим. Тогда из (7) следует, что $Q(x)|x^{q^n} - x$ и $Q(x)$ не делит $x^{q^m} - x$, если m – собственный делитель n (т.е. n не делит m), что доказывает первую половину теоремы.

Допустим, что $R(x)$- неприводимый делитель многочлена *Q(x)*. Тогда $R(x)|(x^{q^n} - x)$и не делит $x^{q^{\frac{n}{p}}} - x$ для любого простого делителя p числа n. Используя (7), имеем: $deg(R)|n$ и не делит $\frac{n}{p}$ для любого простого делителя p числа n. Отсюда следует, что

deg(R)=n и потому $R \sim Q$. Следовательно, *Q(x)* – неприводимый многочлен. Теорема доказана.

2.2.2. Тест Херлестама.

Алгоритм Берлекемпа при всех его достоинствах имеет достаточно серьезный недостаток, что делает его ограниченно применимым для программной реализации проверки многочленов на неприводимость – он медленно работает. По этой причине в программе реализован еще один алгоритм для теста на неприводимость, опубликованный в статье Тора Херлестама (Tore Herlestam). Он значительно более быстрый, чем алгоритм Берлекемпа, но не является универсальным, а зависит от степени тестируемого многочлена.

Определение 1.

1. Аддитивным разложением (*additive decomposition*) числа *n* называется сумма вида:

$$n = d_1 + d_2 + \ldots + d_r, \forall r, d_{i,\underline{i=1,r}} \quad (8)$$

2. Число *n* называется допустимым (*admissible*), если для всех его аддитивных разложений выполняется условие $\text{НОК}(d_1, d_2, \ldots, d_r) < n$.

3. Число *n* называется недопустимым (*non-admissible*) если условие $\text{НОК}(d_1, d_2, \ldots, d_r) = n$ выполняется хотя бы для одного аддитивного разложения.

Лемма 1.

Любое целое число, представляющее собой произведение недопустимого числа на любое целое, является недопустимым.

Доказательство.

Пусть $n = d_1 + d_2 + \ldots + d_r$ и $\text{НОК}(d_1, d_2, \ldots, d_r) = n$, тогда $kn = (kd_1) + (kd_2) + \ldots + (kd_r)$, откуда следует, что $\text{НОК}(kd_1, kd, \ldots, kd_r) = kn$. Лемма доказана.

Теорема 2.

Положительное целое допустимо тогда и только тогда, когда оно – степень простого числа или равно произведению только двух простых чисел.

Теорема 3*. (тест на неприводимость)*

Пусть $f(x) \in F_q[x]$, а число $n = deg(f(x))$ – допустимо. Тогда условия:

1. $f(x_0) \neq 0,\ \forall x_0 \in F_q$,

2. $x^{q^n} = x\ (mod\ \ f(x))$,

3. $x^{q^m} \neq x\ (mod\ f(x)\ ), \forall\ 0 < m < n.$

выполняются тогда и только тогда, когда $f(x)$ – неприводим.

Замечание. *В англоязычной литературе неприводимый нормированный многочлен называется prime polynomial.*

2.3. Тест на примитивность.

Тест на примитивность, используемый данной программой предложен Аланеном (Alanen) и Кнутом (Knuth) и опубликован Аланеном в 1964. Он достаточно прост для понимания и очень быстро работает, что делает его идеально подходящим для использования в описываемой программе. Тест заключается в следующей последовательности проверок, которым подвергается тестируемый многочлен.

1. $f(x_0) \neq 0,\ \forall x_0 \in F_q$,

2. $x^{q^n} = x\ (mod\ \ f(x))$,

3. $x^d \neq 1\ (mod\ f(x)\ ), d = \frac{q^n}{p'}$, где p' – любой простой делитель числа $q^n - 1$.

Если многочлен удовлетворяет всем этим условиям, – он является примитивным.

Замечание. *В англоязычной литературе примитивные многочлены называют как maximum length polynomial, так и primitive polynomial.*

3. Описание реализации программы.

Программа «Euclid» написана на языке C++ (среда Microsoft Visual C++). Причиной выбора языка стало удобство объектно-ориентированного подхода к решению задач подобного рода. Эти преимущества станут понятны далее. Объектно-ориентированный подход предполагает, что все программные элементы представляют собой объекты, являющиеся реализациями (инстанциями) своих классов. При таком подходе все свойства и методы работы с объектами того или иного класса проектируются при создании класса и описываются внутри них. Такая модель прекрасно вписывается в реальное положение вещей, поскольку элемент простого поля (или, что аналогично, многочлен) представляет собой некоторую сущность, над которой можно проводить те или иные действия по определенным правилам. Задачей программиста здесь является

лишь построение однозначного соответствия классов программы описываемым алгебраическим структурам. Программирование было подчинено следующим этапам.

1. Описание класса, соответствующего элементу простого поля (класс *Field*).
2. Описание класса, соответствующего многочлену с элементами из конечного поля (класс *Polynomial*). Очевидно, что многочлен будет представлять собой массив из элементов типа *Field*. Описание прототипов функций классов *Field* и *Polynomial* см. в приложении 1. Описание этих двух классов заключают в себя почти все математические основы проводимых расчетов.
3. Описание вспомогательных классов, используемых при вычислениях, связанных, например, с делением (класс для хранения частного и остатка), алгоритмом Евклида (класс, реализующий структуру *PV*). В этих же вспомогательных классах описан ряд вспомогательных функций, используемых, в различных местах программы, например, функция факторизации степени, возвращающая связный список из простых делителей числа и т.п.
4. Описание классов интерфейса пользователя и средств диагностики возможных ошибок. Здесь разрабатывались все диалоговые окна, которые пользователь может видеть при использовании программы, а также всевозможные обработчики ошибок.
5. Описание вспомогательных классов, связанных, например, с многонитевым (multi-thread) программированием, для параллельной обработки интерфейсных событий и проведения вычислений.
6. Тестирование и написание документации.

Автор не считает необходимым построчное объяснение работы каждой функции, поскольку объем исходных текстов составляет 101Кб, к тому же исходные тексты свободно доступны и снабжены исчерпывающими комментариями.

Приложение 1.

В приложении приведем описание основных классов программы: *Polynomial* и *Field*.

1. Класс **Field**.

```
/* Класс поля элементов по простому модулю с операциями в поле */
class Field
{
public:
   long Module;//модуль
   long Data;//ячейка для хранения числа
public:
   Field();
   Field(long m, long data);
   Field(Field &);
   CString toString();//строковое представление
   Field &operator +(Field &);
   Field &operator -(Field &);
   Field &operator *(Field &);
   Field &operator /(Field &);
   Field &field_pow(int);//возведение в целую степень
   void operator =(Field &);
   int operator ==(Field &);
   void setData(long);
   void setModule(long);
   void set(long,long);

   virtual ~Field();

};
```

2. Класс **Polynomial.**

```
/* Класс реализующий полином с операциями над ним */
class Polynomial
{
public:
   Polynomial(char *coeff,int len, long mod, BOOL endian = FALSE);
   Polynomial(int deg = 0, Field *coeff = NULL);
   void set(int deg = 0, Field *coeff = NULL);
   Polynomial(Polynomial &);
   Polynomial &operator +(Polynomial &);
   Polynomial &operator -(Polynomial &);
   Polynomial &operator *(Polynomial &);
   void operator =(Polynomial &);
   Div_poly &div(Polynomial &p, CProgressCtrl *progress = NULL);
   int operator ==(Polynomial &);
   CString toString(BOOL endian = FALSE);
   virtual ~Polynomial();
   void del_fict_coeff();//удаляет нулевые коефф. при старших
                         //степенях
   void del_fict_coeff(int &deg, Field * & coeff);
   int   isSimple(CProgressCtrl   *progress   =   NULL);//проверяет   на
неприводимость
   int isSimpleB(CProgressCtrl *progress = NULL);//тест Берлекемпа
   int isPrime(CProgressCtrl *progress = NULL);//проверяет на примитивность
   Field &calculate(Field &);//вычисляет значение многочлена
                             //на каком-лидо элементе поля
   static fac_rec* factor(long);//осуществляет факторизацию
   static BOOL isAdmissible(int deg);//проверяет степень на допустимость
   static void kill_factor(fac_rec *);
   static void endian_hdl(char *,int,BOOL);//преобразует строку из
```

```
         //little-endian в big-endian формат и наоборот
public:
   int Deg;//степень
   int Err;//код ошибки
   Field *Coeff;//массив коэффициентов
   //считает общее число объектов этого класса
   static int counter;
   //лог куда пишутся все преобразования
   static FILE *log;
private:
   int was_new;
private:
   void add_coeff(int);//добавляет нулевые коефф.
                           //до указанной степени
   int make_compare(Polynomial &);//делает сравнимыми
                           //при помощи add_coeff(int)
};
```

Приложение 2. Руководство пользователя.

Руководство пользователя состоит из трех разделов. В первом разделе даются общие представления о программе и ее работе. Второй раздел посвящен описанию интерфейса пользователя. В третьем разделе содержатся результаты тестирования программы.

Литература.

1. П. Ноден, К. Китте. Алгебраическая алгоритмика. М: «Мир», 1999г, 720с.
2. Tore Herlestam. On using prime polynomials in crypto generators. Department of Computer Engineering University of Lund. Sweden.
3. Герберт Шилдт. C/C++. Справочник программиста. М: «Вильямс», 2000, 448с.