УДК 681.3.06

Д. И. ЛАВРИНЕНКО

ОПТИМАЛЬНО РАСШИРЕННЫЕ ПОЛЯ В АЛГОРИТМАХ ДЛЯ ЭЛЛИПТИЧЕСКИХ КРИВЫХ

Введение

Математические вычисления в конечных полях (полях Галуа) являются неотъемлемой частью всех криптографических систем, использующих несимметричные алгоритмы шифрования [1]. К таким системам относятся, в том числе, системы, основанные на решении дискретного логарифма [2], на эллиптических [3] и гиперэллиптических кривых [4]. К таким системам предъявляются жесткие требования по производительности, которые напрямую зависят от возможности быстро выполнять математические преобразования в полях Галуа. Особенностью таких систем является то, что элементами этих полей являются числа большой разрядности, которая в несколько раз превышает разрядность современных ЭВМ. Базовой и, следовательно, наиболее важной операцией в таких системах является операция модульного умножения. Следовательно, уменьшение вычислительной сложности данной операции приводит к пропорциональному росту производительности криптографической системы в целом.

В дальнейшем в статье внимание будет сконцентрировано на алгоритмах эллиптических кривых, как наиболее перспективных с точки зрения применения подхода, описанного в статье. Но ничего не мешает применить данный подход и методы в любых других криптографических алгоритмах, использующих вычисления в полях Галуа.

1. Проблемная область и существующие решения

Общий вид записи поля Галуа, в котором производятся преобразования, имеет вид $GF(p^m)$, где p – простое, а m – положительное целое. В криптографии обычно используются частные специальные случаи таких полей с целью уменьшения вычислительной сложности модульных преобразований в таких полях.

Случай $p = 2$ особенно привлекателен с точки зрения аппаратной реализации умножения в поле Галуа, поскольку элементы поля могут быть однозначно представлены сигналами логических «0» и «1». Однако с точки зрения программной реализации умножения случай $GF(2^m)$ не так привлекателен, как кажется на первый взгляд. Причина в том, что современные вычислительные системы оперируют числами большей разрядности, известными как слова. Для достаточно больших m, которые требуются при проектировании реальных криптографических систем, реализация умножения будет иметь большую вычислительную сложность. Существуют методы [3], позволяющие заметно повысить производительность операции умножения в полях $GF(2^m)$. Большинство из них основано на приведении $GF(2^m)$ к виду $GF((2^n)^m)$, где n максимально приближено к разрядности вычислительной сетки ЭВМ. Это позволяет разрешить вышеуказанную проблему, но здесь снижение вычислительной сложности происходит за счёт увеличения сложности пространственной. К тому же достигнутые результаты проигрывают другим существующим в настоящее время методам.

Аналогично, случай $GF(p)$ также имеет трудности реализации на современных ЭВМ. Для представления элементов поля в разрядной сетке ЭВМ необходимо несколько машинных слов, и здесь имеются две основные трудности при реализации операции умножения. Во-первых, при выполнении собственно умножения необходимо учитывать переносы между отдельными словами. Во-вторых, после умножения нужно выполнить модульное преобразование, которое имеет те же трудности реализации и такую же, если не большую, вычислительную сложность. В настоящее время существуют достаточно эффективные методы решения этих проблем, например, модульное преобразование Монтгомери [6]. Другой метод решения проблемы – подбор p специального вида для уменьшения вычислительной сложности собственно операции умножения, как описано в [7].

2. Оптимально расширенное поле и операции в нём

Объединим все вышеуказанные методы оптимизации с целью достижения максимальной производительности в $GF(p^m)$ реализации эллиптических и гиперэллиптических криптосистем. Чтобы

оптимизировать арифметику в поле, ограничим выбор параметров поля p и m следующими требованиями:

1. Выберем p возможно большим, но таким, чтобы оно помещалось в разрядную сетку машинного слова. Это требование позволит с максимальной эффективностью использовать быстрые процессорные команды.

2. Выберем p таким, чтобы оно было псевдопростым числом Мерсенна, то есть вида $2^n \pm c$, при $\log_2 c \le \frac{1}{2} n$. Это позволит выполнять операцию модульного преобразования над элементами поля с минимальной вычислительной сложностью.

3. Выберем m таким, чтобы в поле существовал несократимый бином вида $x^m - \omega$ для эффективного модульного преобразования в расширенном поле. Степень расширенности m тем меньше, чем большим может быть выбрано p.

Поле, выбранное таким образом, назовём оптимально расширенным полем (ОРП). Операции умножения в таком поле, как уже было указано ранее, позволяют совместно применять виды оптимизации, описанные в [7].

Для такого поля $GF(p)$ является подполем, m – степенью расширения, так что поле можно обозначить как $GF(p^m)$. Это поле является изоморфным по отношению к $GF(p)[x]/(P(x))$, где $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i$, $p_i \in GF(p)$ является нормированным несократимым полиномом степени m над $GF(p)$. Элемент поля $A \in GF(p^m)$ в каноническом (полиномиальном) виде можно записать как

$$A(x) = a_{m-1} x^{m-1} + \ldots + a_1 x + a_0,$$

где $a_i \in GF(p^m)$. Так как p выбиралось таким образом, чтобы не превышать разрядность машинного слова, то $A(x)$ можно представить с помощью m регистров или машинных слов.

Все математические преобразования в поле выполняются по модулю полинома поля. Выбор полинома поля определяет сложность операции модульного преобразования. В дальнейшем здесь будут описываться только операции сложения, умножения и возведения в квадрат и оценена их вычислительная сложность. Введём следующие обозначения для определения вычислительной сложности операций в подполе и расширенном поле:

I_{a0} – вычислительная сложность машинной команды сложения, вычитания или пересылки,

I_{l0} – вычислительная сложность машинной команды цикла или перехода,

I_{m0} – вычислительная сложность машинной команды умножения.

2.1 Сложение и вычитание

Сложение или вычитание двух элементов поля реализуется как последовательное сложение или вычитание коэффициентов соответствующих степеней полинома и, если необходимо, выполнение модульного преобразования по модулю p. Если сравнивать данную реализацию по отношению к операции сложения или вычитания в $GF(p)$, то можно заметить, что здесь не требуется учитывать переносы между машинными словами в процессе вычисления. Это является небольшим преимуществом по отношению к $GF(p)$. Приведём схему алгоритма сложения (схема алгоритма вычитания отличается незначительно и имеет аналогичную вычислительную сложность):

Алгоритм 1. Сложение в расширенном поле $GF(p^m)$

Исходные данные: $A(x) = a_{m-1} x^{m-1} + \ldots + a_1 x + a_0$, $B(x) = b_{m-1} x^{m-1} + \ldots + b_1 x + b_0$, $A(x), B(x) \in GF(p^m)$

Результат: $A(x) + B(x) = C(x) \in GF(p^m)$

for $i = 0$ to $m - 1$
$c_i = a_i + b_i$
if $c_i \geq p$
$c_i = c_i - p$
end if
end for

Вычислительная сложность алгоритма составит:

$$I_{addOEF} = m \cdot (I_{a0} + I_{l0} + \frac{1}{2} I_{a0}) = m \cdot I_{addsf} \quad (1)$$

2.2 Умножение

Операция умножения в ОРП выполняется в два этапа: собственно умножение и модульное преобразование. Так как операция умножения в поле является базовой операцией и, следовательно, самой критичной по вычислительной сложности для криптосистем с открытым ключом, то в дальнейшем мы уделим максимальное внимание всем аспектам реализации данной операции.

Первый этап представляет собой обычное полиномиальное умножение:

$$C'(x) = A(x) \times B(x) = c'_{2m-2} x^{2m-2} + \ldots + c'_1 x + c'_0 ; c'_i \in GF(p)$$

Данный метод требует m^2 умножений и $(m-1)^2$ сложений в подполе $GF(p)$. Следовательно, вычислительная сложность данной операции составит:

$$I_{mul} = I_{mulsf} \cdot m^2 + I_{addsf} \cdot (m-1)^2, \quad (2)$$

где I_{mulsf} и I_{addsf} обозначают вычислительную сложность умножения и сложения в подполе $GF(p)$ соответственно. Операция умножения в подполе $GF(p)$ является здесь ключевой и будет подробно рассмотрена ниже, в разделе 2.2.1.

Второй этап – модульное преобразование в поле $GF(p^m)$:

$$C(x) = C'(x) \bmod P(x); C(x) \in GF(p^m)$$

Обозначим вычислительную сложность данного этапа как I_{red}. Ниже, в разделе 2.2.2, будет представлен эффективный алгоритм для выполнения такого преобразования и оценена его вычислительная сложность.

Вычислительная сложность операции умножения в ОРП будет равна сумме вычислительных сложностей операций умножения и модульного преобразования, т.е.

$$I_{mulOEF} = I_{mul} + I_{red} \quad (3)$$

2.2.1 Умножение в подполе *GF(p)*

Как уже было упомянуто выше, производительность операции умножения в подполе $GF(p)$ является одним из определяющих факторов для производительности умножения в поле $GF(p^m)$.

Умножение двух элементов $a, b \in GF(p)$ выполняется как $a \times b \bmod p$. Благодаря тому, что разрядность элемента $GF(p)$ не превышает разрядности машинного слова, умножение можно выполнить непосредственно одной машинной инструкцией. Например, для 32-разрядных машин наибольшим простым числом, которое помещается в разрядную сетку, является $2^{32} - 5$. Следовательно. для такой ЭВМ нужно выбирать $p \leq 2^{32} - 5$. Подытоживая, можно рекомендовать использование p как можно большей величины, пока оно остаётся в пределах разрядной сетки.

При умножении двух целых чисел размером в машинное слово результатом, в общем случае, является целое размером в двойное машинное слово. Чтобы завершить вычисление результата, нужно

выполнить модульное преобразование. Тривиальное решение этой проблемы – получение остатка при делении двух целых чисел. Однако инструкция деления в современных ЭВМ имеет самую большую вычислительную сложность среди арифметических инструкций. Выбор параметра p в соответствии с требованием 2 (см. раздел 2) позволяет применить гораздо более эффективный алгоритм.

Хорошо известно, что возможно быстрое модульное преобразование по модулю вида $2^n - c$, где c является «малым» целым числом. Модуль такого вида позволяет выполнить модульное преобразование без операции деления. Здесь будет представлен алгоритм, базирующийся на алгоритме из [9]. Здесь будут рассматриваться только модули вида $2^n - c$, хотя незначительная модификация алгоритма позволяет использовать вид $2^n + c$. В приведённом ниже алгоритме операторы << и >> обозначают операции сдвигов влево и вправо соответственно.

Алгоритм 2. Модульное преобразование в подполе $GF(p)$

Исходные данные: $p = 2^n - c, \log_2 c \le \frac{1}{2} n,\ x < p^2$

Результат: $r \equiv x \bmod p$

$q = x >> n$
$res = x - q << n$
$r = res$
while $q > 0$
$t = qc$
$q = t >> n$
$res = t - q << n$
$r = r + res$
end while
while $r \ge p$
$r = r - p$
end while

В соответствии с выбранными требованиями главный цикл алгоритма будет выполнен, как максимум, два раза, т.е. вычислительная сложность алгоритма составит:

$$I_{redsf} = 2(I_{m0} + I_{l0}) + 12I_{a0} \tag{4}$$

Если n в точности соответствует разрядности машинного слова (а в практической реализации такой случай будет преимущественно использоваться), то операции сдвига можно выполнить простым присвоением, что позволит дополнительно снизить вычислительную сложность:

$$I_{redsf} = 2(I_{m0} + I_{l0}) + 6I_{a0} \tag{4'}$$

Полная вычислительная сложность умножения в подполе составляет:

$$I_{mulsf} = I_{m0} + I_{redsf} \tag{5}$$

При практической реализации данного алгоритма получается значительный выигрыш в производительности по отношению к выполнению непосредственного деления.

2.2.2 Модульное преобразование в расширенном поле $GF(p^m)$

После выполнения умножения элементов поля $GF(p^m)$ в полиномиальном представлении мы получаем промежуточный результат $C'(x)$. В общем случае степень $C'(x)$ больше или равна m. В этом случае требуется выполнить модульное преобразование. Канонический метод выполнения такого преобразования – полиномиальное деление с остатком на полином поля $P(x)$. Вычислительная сложность этого преобразования является одним из самых высоких среди операций арифметики многократной точности [6], и она тем выше, чем больше число членов в полиноме поля. Но если использовать полиномы специального вида, в частности, с малым количеством членов и малым весом коэф-

фициентов, то вычислительную сложность можно значительно снизить. Это обеспечивается требованием 3 (см. раздел 2) о том, чтобы полином поля имел вид:

$$P(x) = x^m - \omega .$$

По определению, $C'(x)$ имеет вид:

$$C'(x) = c'_{2m-2}\, x^{2m-2} + \ldots + c'_m\, x^m + c'_{m-1}\, x^{m-1} + \ldots + c'_1\, x + c'_0$$

Только члены $c'_{m+i}\, x^{m+i}, i \geq 0$ должны быть преобразованы по модулю $P(x)$. Заметим, что

$$c'_{m+i}\, x^{m+i} \equiv \omega c'_{m+i}\, x^i \bmod P(x);\ i = 0,1,\ldots,m-2$$

Так как степень $C'(x) \leq 2m-2$, требуется максимум $m-1$ умножения на ω и $m-1$ операций сложения для редуцированных членов.

Общий вид выражения полиномиального преобразования по модулю:

$$C(x) \equiv c'_{m-1}\, x^{m-1} + [\omega c'_{2m-2} + c'_{m-2}]x^{2m-2} \ldots + [\omega c'_{m+1} + c'_1]x + [\omega c'_m + c'_0] \bmod P(x)$$

Вычислительная сложность преобразования составляет:

$$I_{red} = (m-1)(I_{mulsf} + I_{addsf}) \qquad (6)$$

2.3 Возведение в квадрат

Операцию возведения в квадрат можно выполнить, используя метод операции умножения. Но хорошо известен метод, позволяющий сократить количество требуемых операций умножения в подполе почти в два раза. Этот метод требует $m(m+1)/2$ операций умножения в подполе вместо m^2 в операции умножения. Пропорционально уменьшается также и число операций сложения в подполе, а модульное преобразование будет точно таким же, как и для операции умножения. Следовательно, вычислительная сложность данной операции составит:

$$I_{sqrOEF} = (m(m-1)/2)(I_{mulsf} + I_{addsf}) + I_{res} \qquad (7)$$

2.4 Дополнительная оптимизация и улучшения

Обратим внимание на то, что два особых частных случая предоставляют возможность дополнительной оптимизации. Назовём их типами ОРП I и II.

Тип I имеет p вида $2^n \pm 1$. Такой случай позволяет выполнять модульное преобразование над элементами подполя с минимальной вычислительной сложностью. Тогда главный цикл алгоритма 2 будет выполняться только единожды, и в цикле не требуются операции умножения. В этом случае вычислительная сложность модульного преобразования в подполе будет равна

$$I_{redsfI} = 8I_{a0}, \qquad (8)$$

а вычислительная сложность всей операции умножения в расширенном поле

$$I_{mulOEFI} = I_{mul} + I_{red} = (m^2 + m - 1)I_{m0} + (9.5m^2 + 6.5m - 8)I_{a0} \qquad (9)$$

Тип II имеет несократимый бином вида $x^m - 2$. Полином такого вида позволяет ускорить модульное преобразование в расширенном поле. В этом случае операции умножения на ω можно заменить на операции сдвигов. Вычислительные сложности модульного преобразования и умножения в расширенном поле будут равны соответственно:

$$I_{redII} = (m-1)(\frac{3}{2}I_{a0} + I_{addsf}) \qquad (10)$$

$$I_{mulOEFII} = 2m^2 I_{m0} + (7{,}5m^2 - 1.5)I_{a0} \qquad (11)$$

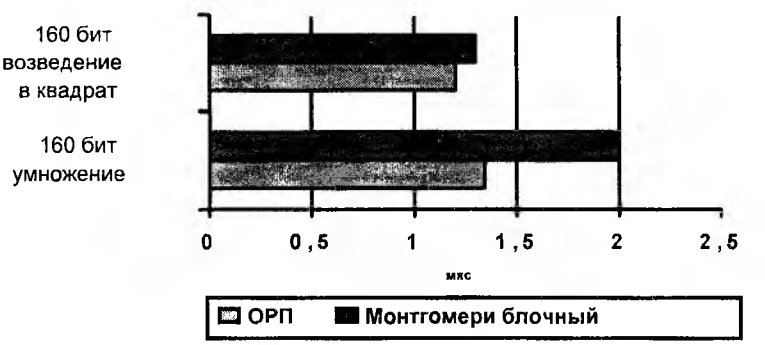
3 Экспериментальные результаты

Для получения экспериментальных результатов реализуем алгоритмы на наиболее распространённой в настоящее время архитектуре процессоров Intel Pentium II. Поскольку разрядность процессора – 32 бита, то выбор параметров был произведен следующим образом: $p = 2^{32} - 5$, $m = 5$, $\omega = 2$. Данный случай соответствует ОРП II и позволяет применить дополнительную оптимизацию.

Производительность операций умножения и возведения в квадрат в ОРП будем сравнивать с производительностью аналогичных операций в $GF(p)$, $p \le 2^{160}$ с применением блочной арифметики Монтгомери [6] как с наиболее производительной на текущий момент и аналогичной по криптостойкости. Теоретический расчёт вычислительной сложности будем производить, установив $I_{m0} = 4$, $I_{l0} = 2$, $I_{a0} = 1$. С расчётом вычислительной сложности операций в $GF(p)$ подробно можно ознакомиться в [6].

Вместе с теоретическим расчётом проведём вычислительный эксперимент оценки вычислительной сложности. Результаты были получены на ЭВМ с процессором Pentium II 366 MHz. Результаты показаны на графике.

Теоретические и экспериментальные данные сведены в таблицу.



Метод	Выч. сложность, квадрат	Выч. сложность, умножение	Время выполнения, квадрат	Время выполнения, умножение
Блочн. Монтгомери	485 ед.	825 ед.	1,304 мкс	1,983 мкс
ОРП	446 ед.	524 ед.	1.203 мкс	1,346 мкс

Исходя из полученных данных, можно заключить, что метод ОРП обеспечивает более высокую производительность по сравнению с блочной арифметикой Монтгомери.

Заключение

Принимая во внимание экспериментальные результаты арифметики в ОРП, можно рекомендовать данный подход для использования как в криптосистемах на базе эллиптических и гиперэллиптических кривых, так и в любых других, которые используют операции умножения и возведения в степень в полях Галуа. В ОРП можно также применить метод проективных координат [11], который позволит получить дополнительный выигрыш в производительности операции возведения в степень.

В данной статье были рассмотрены исключительно математические аспекты оптимизации математических преобразований в полях Галуа. Но ничто не мешает нам применить совместно с ними и другие методы оптимизации, например, аппаратно-зависимую для конкретного типа ЭВМ.

Список литературы 1. *Diffie W., Hellman* M.E. New directions in cryptography // IEEE Trans. on Information Theory. 1976. V.IT-22. №6. P. 644-654. 2. *Lenstra A.K., Manasse M.S.* Factoring with two large primes // Advances in cryptology – Eurocrypt '90. Berlin,1991. P. 72-82. 3. *R. Schroeppel, H. Orman, S. O'Malley, et. al.* Fast key exchange with elliptic cryptosystems // Advances in Cryptography – CRYPTO '95. 1995. 4. *Menezes A., Oorschot P., Vanstone S.* Handbook of Applied Cryptography. CRC Press, 1996 816 p. 5. *Кнут Д.* Искусство программирования для ЭВМ: В 3-х т. Т.2. Получисленные алгоритмы. М.: Мир, 1977. 387 с. 6. *Свинарёв А.В.* Методы и средства комбинированных несимметричных криптографических преобразований информации с уменьшенной вычислительной сложностью // На правах рукописи. 1998. 7. *Kenji Koyama and Yukio Tsuruoka.* Speeding up elliptic cryptosystems by used a signed binary window method // In Crypto' 92 Springer Lecture Notes in Computer Science. 1992.

Харьковский государственный технический университет радиоэлектроники

Поступила в редколлегию 27.03.2001