УДК 681.3.06:519.248.681

Ю.В. СТАСЕВ, д-р техн. наук, С.А. ГОЛОВАШИЧ, канд. техн. наук, В.Ю. КОВТУН

СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ УМНОЖЕНИЯ И ПРИВЕДЕНИЯ ПО МОДУЛЮ В ПОЛЕ $GF(2^m)$

Введение

Подсистемы криптографической защиты информации стали неотъемлемым элементом широкого спектра современных информационных технологий. Основным требованием к алгоритмам криптозащиты является обеспечение необходимого уровня стойкости, однако возможность практического внедрения этих алгоритмов определяется сложностью эффективной реализации на современных аппаратных платформах. Одним из основных элементов современных систем криптографической защиты информации являются асимметричные алгоритмы, в основе которых лежат практически неразрешимые математические задачи. С ростом производительности современных вычислительных систем, возникает проблема повышения криптостойкости этих алгоритмов. В связи с этим, криптографы всего мира все больше внимания уделяют преобразованиям в группе точек эллиптической кривой. Применение этих преобразований, вместо преобразований в поле простого числа, позволяет строить ассиметричные криптоалгоритмы более высокого класса стойкости, однако, обладающие меньшей производительностью. Поэтому работы посвященные повышению производительности этих алгоритмов являются весьма актуальными.

Для представления координат точек эллиптической кривой используют нормальный либо полиномиальный базис. В работах [1, 2] для программной реализации рекомендуется использовать полиномиальный базис с образующим триномом либо пентаномом, а при аппаратной реализации – нормальный базис.

В этой работе мы сосредоточим наше внимание на арифметике в полиномиальном базисе, и статья будет посвящена сравнительному анализу вычислительной сложности различных алгоритмов умножения в поле $GF\left(2^m\right)$, часто используемых при реализации ассиметричных алгоритмов.

Элементами поля $GF(2^m)$ являются двоичные полиномы степени меньше m. Для описания рассматриваемых алгоритмов будем использовать следующие обозначения: $a(x) = a_{m-1}x^{m-1} + ... + a_2x^2 + a_1x + a_0$, $a(x)$, $b(x)$, $c(x) \in GF(2^m)$, $a_i, b_i, c_i \in GF\left(2\right)$, где m – степень неприводимого полинома $f\left(x\right) = x^m + f_{m-1}x^{m-1} + ... + f_2x^2 + f_1x + f_0$, образующего поле. Каждому двоичному полиному можно поставить в соответствие двоичный вектор $(a_{m-1},...,a_2,a_1,a_0)$ длиной m бит. Все операции над элементами поля $GF\left(2^m\right)$ могут быть реализованы посредством преобразования этих векторов.

Все рассматриваемые далее алгоритмы, используют две базовые операции: сложение двух элементов поля (полиномов) и умножение полинома на x в поле.

Реализация всех рассматриваемых далее операций может быть сведена к суперпозиции управляемых операций сдвига и сложения. При программной реализации этих операций на микропроцессорах с разрядностью регистра ALU равной w, выполнение каждой операции будет требоваться порядка $\lceil m/w \rceil$ соответствующих операций микропроцессора.

Сложение двоичных полиномов выполняется путем сложения по модулю 2 соответствующих коэффициентов двоичных векторов:

$$c\left(x\right) = a\left(x\right) + b\left(x\right) = \left(a_{m-1} \text{ xor } b_{m-1}, \ldots, a_1 \text{ xor } b_1, a_0 \text{ xor } b_0\right) \quad (1)$$

Сложность операции сложения двух двоичных векторов:

$$I_{add} = t \cdot I_{add}^{w}, \quad (2)$$

где

w – разрядность регистров ALU выбранного процессора, I^w_{add} – сложность операции сложения содержимого двух w битных регистров процессора, $t = \lceil m/w \rceil$ – количество слов процессора, необходимых для представления полинома степени $m-1$.

В качестве второй базовой операции будем использовать операцию «модулярного умножения на x», она реализуется следующим образом:

$$c(x) = b(x) \cdot x \bmod f(x)$$
$$(c_{m-1},\ldots,c_0) = (b_{m-2} \text{ xor } (f_{m-1} \text{ and } b_{m-1}),\ldots,b_0 \text{ xor } (f_1 \text{ and } b_{m-1}), b_{m-1}) \quad (3)$$

Таким образом, эта операция может быть реализована посредством одного логического сдвига влево и одного «условного» сложения (если степень $b(x) \cdot x$ станет, равна m, другими словами если $b_{m-1} = 1$). Сложность описанной операции:

$$I_{x \bmod} = t \cdot I^w_{shift} + 0.5 \cdot t \cdot I^w_{add} \quad (4)$$

где

I^w_{shift} – сложность операции сдвига на 1 содержимого w разрядного регистра, с возможностью контроля «теряемого» разряда, I^w_{add} – время выполнения операции сложения содержимого двух w разрядных регистров, 0.5 – вероятность выполнения операции сложения с модулем (будем предполагать, что значения элементов $b_i \in GF(2)$ вектора $b(x)$ – равновероятны).

1. Умножение

Теперь рассмотрим различные алгоритмы выполнения операции умножения элементов поля $GF(2^m)$.

$$c(x) = a(x) * b(x) = a_{m-1}x^{m-1} \cdot b(x) + \ldots + a_1 \cdot b(x) + a_0 \cdot b(x) \bmod f(x) \quad (5)$$

Проведем краткий обзор алгоритмов умножения, которые встречаются в литературе.

В работе [3], приведено несколько алгоритмов умножения. Рассмотрим их.

Алгоритм 1. Справа на лево, со сложением и сдвигом и приведением по модулю.

Вход: $a(x), b(x) \in GF(2^m)$.

Выход: $c(x) = a(x) \cdot b(x) \bmod f(x)$.

1. $c(x) = 0$.
2. For $i = 0$ to $m-1$ do
   2.1. If $a_i = 1$ then $c(x) = c(x) + b(x)$.
   2.2. $b(x) = b(x) \cdot x \bmod f(x)$.
3. Return $c(x)$.

Сложность реализации алгоритма 1:

$$I(A_1) = t \cdot (m \cdot I^w_{shift} + m \cdot I^w_{add} + \lceil 0{,}5 \cdot m \rceil \cdot I^w_{add}) = t \cdot (m \cdot I^w_{shift} + \lceil 1{,}5 \cdot m \rceil \cdot I^w_{add}), \quad (6)$$

Приведенный ниже алгоритм 2, аналогичен алгоритму 1, но не выполняет приведение по модулю на каждой итерации. Этот алгоритм является наиболее очевидным способом реализации операции умножения в поле $GF(2^m)$. Эта операция может быть разбита на две операции – умножение двух элементов поля и приведение результата по модулю образующего полинома, т.к. в большинстве криптографических приложений используются полиномы специального вида, рассмотрим эти две операции по отдельности.

Алгоритм 2. Справа на лево, со сложением и сдвигом.

Вход: $a(x), b(x) \in GF(2^m)$.

Выход: $c(x) = a(x) \cdot b(x)$.

1. $c(x) = 0$.

2. For $i = 0$ to $m-1$ do

2.1. If $a_i = 1$ then $c(x) = c(x) + b(x)$.

2.2. $b(x) = b(x) \cdot x$.

3. Return $c(x)$.

Сложность реализации алгоритма 2:

$$I(A_2) = m \cdot (2 \cdot t \cdot I^w_{shift} + 0{,}5 \cdot 2 \cdot t \cdot I^w_{add}) = t \cdot m \cdot (2 \cdot I^w_{shift} + I^w_{add}), \quad (7)$$

Коэффициент 2 в соотношении (7) обусловлен двукратным превышением длины результата по сравнению с длиной аргументов, т.к. приведение по модулю не выполняется.

Эффективность алгоритма 2, может быть повышена – см. алгоритм 3. В этом алгоритме, выигрыш в производительности достигается за счет хранения предвычисленных значений $u(x) \cdot b(x)$, где $u(x) \in GF(2^q), b(x) \in GF(2^m)$, q – ширина «окна» предвычислений. В алгоритме на каждой итерации цикла 3 рассматривается сразу q бит элемента $a(x)$. Число q желательно выбирать таким, что бы оно являлось делителем ширины машинного слова w.

Алгоритм 3. Справа на лево, со сложением и сдвигом, с шириной окна предвычислений q.

Вход: $a(x), b(x) \in GF(2^m)$.

Выход: $c(x) = a(x) \cdot b(x)$.

1. Вычислить $b_u(x) = u(x) \cdot b(x)$ $u(x) \in GF(2^q)$.

2. $c(x) = 0$.

3. For $i = (m-1)/q$ downto 0 do

3.1. $u(x) = (a_{iq+q-1}, a_{iq+q-2}, \ldots, a_{iq+1}, a_{iq})$.

3.2. $c(x) = c(x) + b_u(x) \cdot x^{iq}$

3.3. $c(x) = c(x) \cdot x^q$.

4. Return $c(x)$.

Сложность реализации алгоритма 3:

$$I(A_3) = \left(2t \cdot \left\lceil \tfrac{m-1}{q} \right\rceil + \left\lceil \tfrac{m-1}{q} - \tfrac{m-1}{w} \right\rceil \cdot \left\lceil \tfrac{m+q}{w} \right\rceil \right) \cdot I^w_{shift} + \left\lceil \tfrac{m-1}{q} \right\rceil \cdot \left\lceil \tfrac{m+q}{w} \right\rceil \cdot I^w_{add}) +$$
$$- \left\lceil \tfrac{m+q}{w} \right\rceil \cdot \left(\left(2^q - q - 3\right) \cdot I^w_{add} + (q-1) \cdot I^w_{shift} \right)_{pre} \quad (8)$$

Оптимизируем алгоритм 2 для программной реализации. Следующий алгоритм, базируется на возможности вычисления значения $x^{wj+k} \cdot b(x)$, $k \in (0, w)$, путем логического сдвига на j машинных слов влево вектора соответствующего $x^k \cdot b(x)$. При этом, сдвиг на величину кратную разрядности машинного слова w, практически не требует вычислительных затрат.

Алгоритм 4. Справа на лево, комбинированный метод.

Вход: $a(x), b(x) \in GF(2^m)$.

Выход: $c(x) = a(x) \cdot b(x)$.

1. $c(x) = 0$.

2. For $k = 0$ to $w-1$ do

2.1. For $j = 0$ to $t-1$ do

If $a_{wj+k} = 1$ then $c(x) = c(x) + b(x) \cdot x^{wj}$.

2.2. If $k < w-1$ then $b(x) = b(x) \cdot x$.

3. Return $c(x)$.

В пункте 2.1 производится сложение двух полиномов $c(x)$ и $b(x)$, первый из них имеет двойную длину (т.е. $2t$), в то время как второй имеет длину $t+1$, т.к. в пункте 2.2, умножение $b(x)$ на x, выполняется не более w раз, т.е. длина результата увеличится максимум на одно машинное слово.

Сложность реализации алгоритма 2:

$$I(A_4) = \lceil 0{,}5 \cdot m \cdot (t+1) \rceil \cdot I^w_{add} + (w-1) \cdot (t+1) \cdot I^w_{shift}. \quad (9)$$

Алгоритм 4 просматривает биты в словах – справа на лево, т.е. выполняет «сканирование» в направлении увеличения индекса. Рассмотрим рассматриваемый в [3, 5] алгоритм 5, который просматривает биты слева на право.

Алгоритм 5. Слева на право, комбинированный метод

Вход: $a(x), b(x) \in GF(2^m)$.

Выход: $c(x) = a(x) \cdot b(x)$.

1. $c(x) = 0$

2. For $k = w-1$ downto 0 do

2.1. For $j = 0$ to $t-1$ do

If $a_{wj+k} = 1$ then $c(x) = c(x) + b(x) \cdot x^{wj}$.

2.2. If $k > 0$ then $c(x) = c(x) \cdot x$.

3. Return $c(x)$.

В пункте 2.1 производится сложение двух полиномов $c(x)$ и $b(x)$, первый из них имеет двойную длину (т.е. $2t$), в то время как второй имеет длину t. С каждой итерацией цикла 2.1 длина $c(x)$ увеличивается на w бит, т.е. на одно машинное слово.

Сложность реализации алгоритма 5:

$$I(A_5) = \lceil 0{,}5 \cdot m \cdot t \rceil \cdot I^w_{add} + (w-1) \cdot 2 \cdot t \cdot I^w_{shift}, \quad (10)$$

Алгоритмы 4 и 5 – оба быстрее алгоритма 2, т.к. используют меньшее количество операций сдвига (умножения на x) на этапе 2.2. В пункте 2.1 алгоритма 5, вектор $b(x)$, длиной t машинных слов, складывается с вектором $c(x)$ – для этого требуется t сложений, в то врем как в алгоритме 4 в пункте 2.1. производится сложение вектора $c(x)$ длиной $2t$ с вектором $b(x)$ длиной $t+1$, т.е. требуется $t+1$ сложение машинных слов. Однако, в пункте 2.2 алгоритма 4 выполняется сдвиг $t+1$ машинного слова, в то время как в алгоритме 5 – $2t$ машинных слов. Алгоритм 5 имеет большую сложность, чем алгоритм 4.

Быстродействие алгоритмов 4 и 5 можно значительно повысить за счет увеличения памяти для хранения предвычисленных значений $u(x) \cdot b(x)$, $u(x) \in GF(2^q), b(x) \in GF(2^m)$. На практике ширину окна предвычислений q, такой, чтобы $w \equiv 0 \bmod q$, при этом необходимо не забывать, что в таком случае необходимо хранить $2^q - 1$ полиномов. В этом случае, на каждом шаге, анализируется q двоичных коэффициентов $a(x)$. Модифицированный алгоритм 4, для окна предвычислений шириной q, представлен алгоритмом 6.

Алгоритм 6. Справа на лево, комбинированный метод с шириной окна предвычислений q.

Вход: $a(x), b(x) \in GF(2^m)$.

Выход: $c(x) = a(x) \cdot b(x)$.

1. Вычислить $b_u(x) \leftarrow u(x) \cdot b(x)$, $u(x) \in GF(2^q)$.

2. $c(x) \leftarrow 0$.

3. For $k = 0$ to $w/q - 1$ do

    3.1. For $j = 0$ to $t - 1$ do

        3.1.1. $u(x) \leftarrow (a_{jw+kq+q-1}, a_{jw+kq+q-2}, \ldots, a_{jw+kq+1}, a_{jw+kq})_2$.

        3.1.2. $c(x) \leftarrow c(x) + b_u(x) \cdot x^{wj+kq}$.

4. Return $c(x)$.

В пункте 3.1.1 алгоритма 6 производится выделение кортежей длиной q бит и обращение к таблице предвычислений, для извлечения из нее произведений $b_u(x)$, которые имеют длину $\lceil (m+q)/w \rceil$ машинных слов. В пункте 3.1.2, производится сложение предвычисленного значения $b_u(x)$ с $c(x)$, при этом, на каждой итерации цикла 3.1, предвычесленное значение сдвигается на j машинных слов, а также на kq бит.

Сложность реализации алгоритма 6:

$$I(A_6) = \left\lceil \frac{m}{q} \right\rceil \cdot \left\lceil \frac{m+2q}{w} \right\rceil \cdot I^w_{add} + \left(\frac{m}{q} - 1 \right) \cdot \left\lceil \frac{m+2q}{w} \right\rceil \cdot I^w_{shift} + \\ + \left[\left\lceil \frac{m+q}{w} \right\rceil \cdot \left((2^q - q - 3) \cdot I^w_{add} + (q-1) \cdot I^w_{shift} \right) \right]_{pre}, \quad (11)$$

Модифицированный алгоритм 5, для ширины окна предвычислений q, представлен алгоритмом 7, предложенный в работе [5], который требует предвычислений и в свою очередь имеет меньшую сложность. Определим таблицу предвычислений для $P_{2^q}[u]$, для $\forall\, u \in GF(2^q)$.

$$P_{2^q}[u](x) = \left(u_{2^q-1} x^{2^q-1} + \cdots + u_1 x + u_0 \right) b(x), \quad ()$$

Тогда произведение $a(x)b(x)$ может быть вычислено как

$$a(x)b(x) = \sum_{i=0}^{t-1} \sum_{j=0}^{w-1} a_{iw+j} x^{iw+j} b(x) = \sum_{j=0}^{w-1} x^j \sum_{i=0}^{t-1} a_{iw+j} x^{iw} b(x) = \\ = \sum_{j=0}^{w/q-1} x^{qj} \sum_{i=0}^{t-1} \left(a_{iw+j+(q-1)} x^{(q-1)} + \cdots + a_{iw+j+1} x + a_{iw+j} \right) x^{iw} b(x) = \\ = \sum_{j=0}^{w/q-1} x^{qj} \left(\sum_{i=0}^{t-1} P_{2^q}[u_{i,j}](x) \right) \quad ()$$

где $u_{i,j} = (a_{iw+j+3}, \ldots, a_{iw+j})_2$.

Алгоритм 7. Слева на право, комбинированный метод с шириной окна предвычислений q.

Вход: $a(x), b(x) \in GF(2^m)$.

Выход: $c(x) = a(x) \cdot b(x)$.

1. Вычислить $b_u(x) = u(x) \cdot b(x)$, $u(x) \in GF(2^q)$.

2. $c(x) = 0$.

3. For $k = w/q - 1$ downto 0 do

3.1. For $j = 0$ to $t - 1$ do

3.1.1. $u(x) = (a_{jw+kq+q-1}, a_{jw+kq+q-2}, \ldots, a_{jw+kq+1}, a_{jw+kq})$.

3.1.2. $c(x) = c(x) + b_u(x) \cdot x^{wj}$.

3.2. If $k > 0$ then $c(x) = c(x) \cdot x^q$.

4. Return $c(x)$.

В алгоритме 7, в пункте 3.1.1 производится выделение кортежей длины q бит и выбор из таблицы предвычисленного произведения $b_u(x)$, которое имеет длину $\lceil (m+q)/w \rceil$ машинных слов. В пункте 3.1.2. выполняется прибавление сдвинутого на j слов предвычисленного значения $b_u(x)$ к накапливаемому результату. В пункте 3.2 производится сдвиг на q бит полинома $c(x)$, длиной – $2t$ машинных слов.

Сложность реализации алгоритма 7:

$$I(A_7) = \left\lceil \frac{m}{q} \right\rceil \cdot \left\lceil \frac{m+q}{w} \right\rceil \cdot I^w_{add} + \left(\frac{w}{q} - 1\right) \cdot 2 \cdot t \cdot I^w_{shift} + \left[\!\left[\left\lceil \frac{m+q}{w} \right\rceil \cdot \left(\left(2^q - q - 3\right) \cdot I^w_{add} + (q-1) \cdot I^w_{shift} \right) \right]\!\right]_{pre}, \qquad (12)$$

На практике ширину окна предвычислений q, такой, чтобы $w \equiv 0 \bmod q$, при этом необходимо не забывать, что в таком случае необходимо хранить $2^q - 1$ полиномов. Снизить затраты на память можно за счет

$$\sum_{i=0}^{7} a_i x^i b(x) = \sum_{i=0}^{3} a_j x^j b(x) + \sum_{i=0}^{3} a_{j+j} x^{j+4} b(x), \qquad ()$$

В работах [3, 4], приводится утверждение, о том, что ширина окна предвычисления равная 4, является оптимальной, т.к. не требует много памяти. При дальнейшем увеличении ширины окна предвычислений рост производительности оказывается незначительным, но приводит к значительным затратам памяти. Также достоинством значения $q = 4$ является кратность длине машинного слова.

В основу алгоритма 8 положен метод, который впервые был описан Карацубой для умножения целых чисел [4]. Пусть, k – четное, тогда для умножения двух двоичных полиномов $a(x)$ и $b(x)$ степени меньше k, представим оба сомножителя в виде двух полиномов степени меньше $k/2$:

$a(x) = A_1(x)x^{k/2} + A_0(x),\ b(x) = B_1(x)x^{k/2} + B_0(x)$.

Тогда произведение

$$a(x) \cdot b(x) = A_1 B_1 x^k + [(A_1 + A_0)(B_1 + B_0) + A_1 B_1 + A_0 B_0] x^{k/2} + A_0 B_0, \qquad (13)$$

может быть получено по трем произведениям полиномов степени меньше $k/2$. Соотношение (11) может быть рекурсивно применено для вычисления произведений частичных полиномов. Алгоритм Карацубы особенно эффективен, когда $k = 2nw$, т.к не требует вычислительных затрат при умножении на x^k и $x^{k/2}$.

Рассмотрим два случая: когда образующий полином $f_1(x) = x^{163} + x^7 + x^6 + x^3 + 1$ и $f_2(x) = x^{233} + x^{74} + 1$. Для случая $m = 163$, мы, во первых, расширим элементы поля a и b, так, чтобы их битовая длина была равна 192 (т.е. выровняем полиномы на 32–битную границу) и воспользуемся методом Карацубы для по–членного умножения полиномов $a(x)$ и $b(x)$. Рекурсивно применим метод Карацубы и сведем произведение $a(x) \cdot b(x)$ к умножению полиномов степени меньшей 32. Последнее умножение проведем с помощью варианта алгоритма 4 или с использованием таблиц.

Рассмотрим количество операций, которые необходимо выполнить при рекурсивном применении алгоритма Карацубы (до тех пор, пока количество машинных слов является четным, рассмотрим частный случай, когда $l=1$):

$$\begin{aligned} I(A_8) &= 5\cdot\tfrac{t}{2}\cdot I^w_{add} + 3\cdot I^{t/2}_{mul} = 5\cdot\tfrac{t}{2}\cdot I^w_{add} + 3\cdot\left(5\cdot\tfrac{t}{4}\cdot I^w_{add} + 3\cdot I^{t/4}_{mul}\right) = \\ &= \cdots = 5\cdot\tfrac{t}{2}\cdot I^w_{add} + 3\cdot\left(5\cdot\tfrac{t}{4}\cdot I^w_{add} + 3\cdot\left(5\cdot\tfrac{t}{8}\cdot I^w_{add} + 3\left(5\cdot\tfrac{t}{16}\cdot I^w_{add} + \cdots + 3\cdot I^w_{mul}\right)\right)\right) = \\ &= 5\cdot\tfrac{t}{2}\cdot I^w_{add} + 3\cdot 5\cdot\tfrac{t}{4}\cdot I^w_{add} + 3^2\cdot 5\cdot\tfrac{t}{8}\cdot I^w_{add} + \cdots + 3^{h-1}\cdot 5\cdot\tfrac{t}{2^h}\cdot I^w_{add} + 3^h\cdot I^w_{mul} = \\ &= 5\cdot t\cdot\sum_{i=1}^{h}\tfrac{3^{i-1}}{2^i} + 3^h\cdot I^w_{mul} = 10\cdot I^w_{add}\cdot\left(3^h-2^h\right) + 3^h\cdot I^w_{mul} \end{aligned} \tag{14}$$

где

5 – количество операций сложения на каждом уровне рекурсии, 3 – количество операций умножения на каждом уровне рекурсии, $h=\log_2\lceil\frac{m}{wl}\rceil$ – показатель степени в выражении $l\cdot 2^h\cdot w\ge m$, где $l\cdot 2^h$ – количество слов процессора, необходимых для представления полинома степени $m-1$, $l=2i+1$ - нечетное число слов процессора для умножения которых, необходимо применять алгоритмы описанные выше.

Рассмотрим частный случай, когда $l=1$, тогда общая сложность алгоритма 8:

$$I(A_8) = 10\cdot I^w_{add}\cdot\left(3^h-2^h\right) + 3^h\cdot I^w_{mul}, \tag{15}$$

Повысить производительность алгоритма 8, можно за счет уменьшения сложности I^w_{mul}, то есть вычисления произведения $c(x)$ на $d(x)$, где $c(x)$ та $d(x)$ – полиномы степени меньшей 8. Тогда, для хранения результатов предвычислений, потребуется двухмерная таблица размером 256*256=65536 двухбайтовых слов, то есть 128 kB. Эта таблица универсальна, поэтому ее можно использовать для вычисления произведения любых полиномов принадлежащих $GF(2^8)$. Модификацию алгоритма 8, использующую таблицу предвычислений обозначим через алгоритм 9, его сложность составляет:

$$I\left(A_9\right) = 10\cdot I^w_{add}\cdot\left(3^h-2^h\right) + 3^h\cdot\left(20\cdot I^w_{add} + 9\cdot I_{tbl}\right), \tag{16}$$

где

20 –операций сложения для умножения двух полиномов степеней меньших w,

9 –операций обращения к таблице для умножения двух полиномов степеней меньших 8,

I_{tbl} – сложность операции обращения к таблице для умножения двух полиномов степеней меньших 8.

Следующим предлагаемым алгоритмом является алгоритм предложенный в работах [4, 5].

Представим один из множителей $a(x)$ в виде $t=\lceil m/w\rceil$ двоичных блоков A_i длины w бит, соответствует длине машинного слова:

$$a(x) = A_{t-1}x^{w(t-1)} + \cdots + A_0 = \sum_{i=0}^{t-1} A_i x^{wi},\ A_i = \left(a_{iw+w-1},\ldots,a_{iw}\right)_2 \in F_{2^w},\ i=\overline{0,t-1}, \tag{17}$$

Определим вектор $P_u(x)$ предвычислений:

$$P_u(x) = u_{s-1}x^{w(s-1)}b(x) + u_{s-2}x^{w(s-2)}b(x) + \cdots + u_1x^w b(x) + u_0 b(x). \tag{18}$$

Следовательно произведение $a(x)\cdot b(x)$ может быть представлено:

$$a(x)\cdot b(x) = \sum_{j=0}^{w-1} 2^j \sum_{i=0}^{t-1} a_{iw+j}x^{wi}\cdot b(x) = \sum_{j-0}^{w-1} x^j P\left[I_j\right](x), \tag{19}$$

где $I_j = \left(a_{(s-1)w+j}, \ldots, a_{w+j}, a_j\right)_2$.

Детальное описание алгоритма представлено ниже.

Алгоритм 10. Алгоритм умножения методом Lim-Lee.

Вход: $a(x), b(x) \in GF\left(2^m\right)$.

Выход: $c(x) = a(x) \cdot b(x)$.

1. For $u = 0$ to $2^t - 1$ do

   1.1. $u(x) \leftarrow \left(u_{s-1}, \ldots, u_1, u_0\right)_2$.

   1.2. $P_u(x) \leftarrow \sum_{i=0}^{t-1} u_i x^{wi} \cdot b(x)$.

2. $s(x) \leftarrow 0$.

3. For $j = w-1$ downto 0 do

   3.1. $s(x) \leftarrow s(x) \cdot x$

   3.2. $u(x) \leftarrow \left(a_{(t-1)w+j}, \ldots, a_{w+j}, a_j\right)_2$.

   3.3. $s(x) \leftarrow s(x) + P_u(x)$.

4. Return $(s(x))$.

Как видно из описания алгоритма, в памяти необходимо хранить $2^t - 1$ полиномов длиной $2m = 2t$ бит. Ширину окна предвычислений можно повысить для случая, когда $b(x)$ - является некоторой константой, но, из практических соображений, обычно $t \le 16$. В таком случае вычисления шага 1 можно выполнить при инициализации системы или хранить в виде массива констант. При этом, возможно повысить производительность алгоритма, если каждому $P_u(x)$ ставить в соответствие число, указывающее количество машинных слов, которые необходимо складывать по модулю 2. Что даст возможность сократить в 2 раза количество операций сложения по модулю 2.

Сложность алгоритма 10 в худшем случае составляет:

$$I\left(A_{10}^{worst}\right) = 2tw\left(I_{shift}^{w} + I_{add}^{w}\right), \tag{20}$$

Сложность алгоритма 10 в среднем случае составляет:

$$I\left(A_{10}^{average}\right) = tw\left(2I_{shift}^{w} + \frac{2^t - 1}{2^t} I_{add}^{w}\right), \tag{21}$$

При этом сложность предвычислений составляет:

$$I\left(A_{10}^{pre}\right) = t2^{t-1}\left(tI_{add}^{w}\right) = t^2 2^{t-1} I_{add}^{w}, \tag{22}$$

В сложности предвычислений не учтены сдвиги, т.к. они производятся на w бит, что означает установить соответствующее значение бита в старшем слове, такая операция не является ресурсоемкой.

2. Приведение по модулю

Рассмотрим алгоритмы приведения по модулю, которые приводятся в литературе [3, 4].

Классический универсальный алгоритм, представлен под номером 11, это алгоритм побитного анализа делимого и соответствующего сложения с ним модуля. Этот алгоритм отличается универсальностью, но является наименее производительным, из приведенных ниже.

Алгоритм 11. Приведение по модулю, побитного анализа делимого.

Вход: $c(x) \in GF\left(2^{2m}\right)$.

Выход: $c(x) \bmod f(x)$.

1. For $i = 2m-2$ downto m do

1.1. If $c_i = 1$ then

1.1.1 $j = \lfloor (i-m)/w \rfloor$, $k = (i-m) - wj$

1.1.2. $c(x) = c(x) + f(x) \cdot x^{wj+k}$

2. Return $c(x) \bmod x^m$.

Приведем количество операций необходимых для выполнения приведения по модулю алгоритмом 11:

$$I(A_{11}) = \lceil 0{,}5 \cdot m \cdot (t+1) \rceil \cdot \left(I^w_{add} + I^w_{shift} \right), \tag{23}$$

Производительность алгоритма 11 можно существенно повысить путем предвычисления сдвигов на $k \in [0, w-1]$ разрядов (в пределах машинного слова). Этот подход, представлен алгоритмом 11. Представим образующий полином в виде: $f(x) = x^m + r(x)$. Тогда $x^i \equiv x^{i-m} \cdot r(x) \bmod f(x)$, $i \geq m$. В случае, когда $m > wd + \deg(r(x))$, $d > 0$ получаем значительный выигрыш в производительности, т.к. приходится работать с меньшим количеством машинных слов.

Приведем таблицу $x^i \equiv x^{i-m} \cdot r(x) \bmod f(x)$, для полинома $f_1(x) = x^{163} + x^7 + x^6 + x^3 + 1$, где $i \in [288, 319]$ причем указанный диапазон изменения i равен $w = 32$.

$$\begin{aligned} x^{319} &\equiv x^{163} + x^{162} + x^{159} + x^{156} \\ &\ldots. \\ x^{289} &\equiv x^{133} + x^{132} + x^{129} + x^{126} \\ x^{288} &\equiv x^{132} + x^{131} + x^{128} + x^{125} \end{aligned} \tag{24}$$

Алгоритм 12. Приведение по модулю, побитного анализа делимого с предвычислениями.

Вход: $c(x) \in GF(2^{2m})$.

Выход: $c(x) \bmod f(x)$.

1. Вычислить $u_k(x) = x^k \cdot r(x)$, $k \in [0, w-1]$, $u_k(x) \in GF(2^{m+w})$.

2. For $i = 2m-2$ downto m do

2.1. If $c_i = 1$ then

$j = (i-m) \text{ div } w$, $k = (i-m) \bmod w$

2.2. $c(x) = c(x) + u_k(x) \cdot x^{wj}$

3. Return $c(x) \bmod x^m$.

Количество операций необходимых для выполнения приведения по модулю алгоритмом 12, определяется соотношением:

$$I(A_{12}) = \lceil 0{,}5 \cdot m \cdot (t+1) \rceil \cdot I^w_{add} + w \cdot (t+1) \cdot I^w_{shift}, \tag{25}$$

Если рассматривать полиномы специального вида, т.е. неприводимые трехчлены и пятичленны, тогда производительность операции приведения по модулю может быть существенно увеличена за счет учета структуры полинома $f(x)$. Рассмотрим алгоритмы приведения по модулю для полиномов $f_1(x) = x^{163} + x^7 + x^6 + x^3 + 1$ и $f_2(x) = x^{233} + x^{74} + 1$.

В алгоритме 13 (приведение по модулю $f_1(x)$) используются сдвиги на 29, 3, 28, 4 бит, эти значения получаются следующим образом: рассматриваются пословно биты каждого

слагаемого в столбцах (16), например первое слагаемое – полином $x^{163}+\cdots+x^{133}+x^{132}$, второе слагаемое – полином $x^{162}+\cdots+x^{132}+x^{131}$, третье слагаемое – полином $x^{159}+\cdots+x^{129}+x^{128}$ и четвертое слагаемое – полином $x^{156}+\cdots+x^{126}+x^{125}$, каждое слагаемое длинной в одно машинное слово (32 бита). Для сложения с ними маски $c[i]$, ее приходиться сдвигать влево на $m \bmod w$ бит. Первое слагаемое занимает старшую и младшую части двух машинных слов, поэтому маску выравнивают (сдвигают вправо на 29 бита) перед сложением с первым словом, а потом выравнивают (сдвигают влево на 3 бита) перед сложением со вторым словом. Аналогично для второго и четвертого слагаемых, третье – выровнено изначально. В пункте 2 производится очистка 2, 1, 0 бит, т.к. они не должны влиять на результат, они соответствуют полиномам со степенями $5*w+i$, где $i=\overline{0,2}$ которые имеют степень меньше 163. В пункте 5 производится очистка старших бит делимого, которые соответствуют полиномам со степенями $5*w+i$, где $i=\overline{3,31}$, но они уже были учтены при вычислениях. Приведенный алгоритм.

Алгоритм 13. Приведение по модулю, фиксированного полинома $f_1(x)=x^{163}+x^7+x^6+x^3+1$.

Вход: $c(x)\in GF(2^{2m})$, $m=163$.

Выход: $c(x) \bmod f(x)$.

1. For $i=10$ downto 6 do
   - 1.1. $t=c[i]$
   - 1.2. $c[i-6]=c[i-6]$ xor $(t<<29)$
   - 1.3. $c[i-5]=c[i-5]$ xor $(t<<4)$ xor $(t<<3)$ xor t xor $(t>>3)$
   - 1.4. $c[i-4]=c[i-4]$ xor $(t>>28)$ xor $(t>>29)$
2. $t=c[5]$ and 0xFFFFFFF8
3. $c[0]=c[0]$ xor $(t<<4)$ xor $(t<<3)$ xor t xor $(t>>3)$
4. $c[1]=c[1]$ xor $(t>>28)$ xor $(t>>29)$
5. $c[5]=c[5]$ and 0x00000007
6. Return $c(x) \bmod x^m$.

Общее количество операций необходимых для выполнения приведения по модулю алгоритмом 13:

$$I(A_{13})=41\cdot I^w_{add}+35\cdot I^w_{shift}+2\cdot I^w_{and}+23\cdot I^w_{let}, \qquad (26)$$

где

I^w_{and} – сложность операции логического умножения двух w разрядных слов, I^w_{let} – сложность операции присвоения.

Аналогичный алгоритм, можно привести и для полинома $f_2(x)=x^{233}+x^{74}+1$.

Алгоритм 14. Приведение по модулю, фиксированного полинома $f_2(x)=x^{233}+x^{74}+1$.

Вход: $c(x)\in GF(2^{2m})$, $m=233$.

Выход: $c(x) \bmod f(x)$.

1. For $i=14$ downto 8 do
   - 1.1. $t=c[i]$
   - 1.2. $c[i-8]=c[i-8]$ xor $(t<<23)$
   - 1.3. $c[i-7]=c[i-7]$ xor $(t>>9)$
   - 1.4. $c[i-5]=c[i-5]$ xor $(t<<1)$

1.5. $c[i-4] = c[i-4]$ xor $(t >> 31)$

2. $t = c[7]$ and 0xFFFFFE08
3. $c[0] = c[0]$ xor $(t >> 9)$
4. $c[2] = c[2]$ xor $(t << 1)$
5. $c[3] = c[3]$ xor $(t >> 31)$
6. $c[5] = c[5]$ and 0x000001FF
7. Return $c(x) \bmod x^m$.

Общее количество операций необходимых для выполнения приведения по модулю алгоритмом 14:

$$I(A_{14}) = 31 \cdot I^w_{add} + 31 \cdot I^w_{shift} + 2 \cdot I^w_{and} + 40 \cdot I^w_{let}, \quad (27)$$

Сложность алгоритмов умножения в поле $GF(2^m)$ представлена в таблице 1.

Таблица 1

Название алгоритма	Сложность
Сложение	$I_{add} = t \cdot I^w_{add}$
Алгоритм 2. Справа на лево, со сложением и сдвигом.	$I(A_2) = 2 \cdot t \cdot m \cdot (I^w_{shift} + 0{,}5 \cdot I^w_{add})$
Алгоритм 3. Справа на лево, со сложением и сдвигом, с шириной окна предвычислений q	$I(A_3) = \left(2t \cdot \left\lvert \frac{m-1}{q} \right\rvert + \left\lvert \frac{m-1}{q} - \frac{m-1}{w} \right\rvert \cdot \left\lvert \frac{m+q}{w} \right\rvert \right) \cdot I^w_{shift} + \left\lvert \frac{m-1}{q} \right\rvert \cdot \left\lvert \frac{m+q}{w} \right\rvert \cdot I^w_{add}) + + \left\lceil \frac{m+q}{w} \right\rceil \cdot \left((2^q - q - 3) \cdot I^w_{add} + (q-1) \cdot I^w_{shift} \right) \Big]_{pre}$
Алгоритм 4. Справа на лево, комбинированный метод	$I(A_4) = 0{,}5 \cdot m \cdot (t+1) \cdot I^w_{add} + (w-1) \cdot (t+1) \cdot I^w_{shift}$
Алгоритм 5. Слева на право, комбинированный метод	$I(A_5) = 0{,}5 \cdot m \cdot t \cdot I^w_{add} + (w-1) \cdot 2 \cdot t \cdot I^w_{shift}$
Алгоритм 6. Справа на лево, комбинированный метод с шириной окна предвычислений q.	$I(A_6) = \left\lceil \frac{m}{q} \right\rceil \cdot \left\lceil \frac{m+2q}{w} \right\rceil \cdot I^w_{add} + \left(\frac{m}{q} - 1 \right) \cdot \left\lceil \frac{m+2q}{w} \right\rceil \cdot I^w_{shift} + + \left\lceil \frac{m+q}{w} \right\rceil \cdot \left((2^q - q - 3) \cdot I^w_{add} + (q-1) \cdot I^w_{shift} \right) \Big]_{pre}$
Алгоритм 7. Слева на право, комбинированный метод с шириной окна предвычислений q.	$I(A_7) = \left\lceil \frac{m}{q} \right\rceil \cdot \left\lceil \frac{m+q}{w} \right\rceil \cdot I^w_{add} + \left(\frac{m}{q} - 1 \right) \cdot 2 \cdot t \cdot I^w_{shift} + + \left\lceil \frac{m+q}{w} \right\rceil \cdot \left((2^q - q - 3) \cdot I^w_{add} + (q-1) \cdot I^w_{shift} \right) \Big]_{pre}$
Алгоритм 8. Алгоритм Карацубы	$I(A_8) = 10 \cdot I^w_{add} \cdot \left(3^h - 2^h\right) + 3^h \cdot I^w_{mul}$
Алгоритм 9. Алгоритм Карацубы с предвычислениями	$I(A_9) = 10 \cdot I^w_{add} \cdot \left(3^h - 2^h\right) + 3^h \cdot \left(20 \cdot I^w_{add} + 9 \cdot I_{tbl}\right)$
Алгоритм 10. Алгоритм Lim-Lee	$I\left(A^{average}_{10}\right) = tw\left(2I^w_{shift} + \frac{2^t - 1}{2^t} I^w_{add}\right),\ I\left(A^{pre}_{10}\right) = t^2 2^{t-1} I^w_{add}$

где

m – степень полевого полинома, q – ширина окна предвычислений, I^w_{mul} – сложность операции умножения по модулю 2, двух w – разрядных слов, I^w_{add} – сложность операции, содержимого двух w – разрядных слов, I^w_{shift} – сложность операции сдвига на 1 одного w – разрядного слова, I_{tbl} – сложность операции обращения к таблице, $t = \lceil m/w \rceil$ – количество слов процессора, необходимых для представления полинома степени $m-1$, $h = \log_2 \lceil m/w \rceil$ – показатель степени в выражении $t = 2^h$, где $t = 2^h$ – количество слов процессора, необходимых для представления полинома степени $m-1$.

В таблице 2 приведем расчет сложностей алгоритмов перечисленных в таблице 1, для степени полиномов $m = 256$, разрядности машинного слова $w = 32$.

Таблица 2

Название алгоритма	Сложность
Сложение	$I_{add} = 8 \cdot I^w_{add}$
Алгоритм 2. Справа на лево, со сложением и сдвигом.	$I(A_2) = 4096 \cdot I^w_{shift} + 2048 \cdot I^w_{add}$
Алгоритм 3. Справа на лево, со сложением и сдвигом, с шириной окна предвычислений q	$I(A_3) = 1528 \cdot I^w_{shift} + 576 \cdot I^w_{add} + \left[81 \cdot I^w_{add} + 27 \cdot I^w_{shift}\right]_{pre} = 1555 \cdot I^w_{shift} + 657 \cdot I^w_{add}$
Алгоритм 4. Справа на лево, комбинированный метод	$I(A_4) = 1152 \cdot I^w_{add} + 279 \cdot I^w_{shift}$
Алгоритм 5. Слева на право, комбинированный метод	$I(A_5) = 1152 \cdot I^w_{add} + 496 \cdot I^w_{shift}$
Алгоритм 6. Справа на лево, комбинированный метод с шириной окна предвычислений $q = 4$.	$I(A_6) = 576 \cdot I^w_{add} + 567 \cdot I^w_{shift} + + \left[81 \cdot I^w_{add} + 27 \cdot I^w_{shift}\right]_{pre} = 657 \cdot I^w_{add} + 594 \cdot I^w_{shift}$
Алгоритм 7. Слева на право, комбинированный метод с шириной окна предвычислений $q = 4$.	$I(A_7) = 576 \cdot I^w_{add} + 112 \cdot I^w_{shift} + + \left[81 \cdot I^w_{add} + 27 \cdot I^w_{shift}\right]_{pre} = 657 \cdot I^w_{add} + 139 \cdot I^w_{shift}$
Алгоритм 8. Алгоритм Карацубы	$I(A_8) = 95 \cdot I^w_{add} + 27 \cdot I^w_{mul}$
Алгоритм 9. Алгоритм Карацубы с предвычислениями	$I(A_9) = 635 \cdot I^w_{add} + 243 \cdot I_{tbl}$
Алгоритм 10. Алгоритм Lim-Lee	$I\left(A_{10}^{average}\right) = 256\left(2I^w_{shift} + \frac{255}{256} I^w_{add}\right) = 255 I^w_{add} + 512 I^w_{shift}$, $I\left(A_{10}^{pre}\right) = 64 \cdot 128 I^w_{add} = 8192 I^w_{add}$

В таблице 3 приведем общие сложности универсальных алгоритмов приведения по модулю.

Таблица 3

Название алгоритма	Сложность
Алгоритм 10. Приведение по модулю, побитный анализ делимого.	$I(A_{10}) = \lceil 0{,}5 \cdot m \cdot (t+1) \rceil \cdot \left(I^w_{add} + I^w_{shift}\right)$
Алгоритм 11. Приведение по модулю, побитный анализ делимого с предвычислениями.	$I(A_{11}) = \lceil 0{,}5 \cdot m \cdot (t+1) \rceil \cdot I^w_{add} + w \cdot (t+1) \cdot I^w_{shift}$

В таблице 4 приведем сложность алгоритмов приведения по модулю для случая $m = 163$, $w = 32$.

Таблица 4

Название алгоритма	Сложность
Алгоритм 10. Приведение по модулю, побитного анализа делимого.	$I(A_{10}) = 567 \cdot \left(I^w_{add} + I^w_{shift}\right)$
Алгоритм 11. Приведение по модулю, побитного анализа делимого с предвычислениями.	$I(A_{11}) = 567 \cdot I^w_{add} + 224 \cdot I^w_{shift}$
Алгоритм 12. Приведение по фиксированному модулю	$I(A_{12}) = 41 \cdot I^w_{add} + 35 \cdot I^w_{shift} + 2 \cdot I^w_{and} + 23 \cdot I^w_{let}$

В таблице 5, приведены временные характеристики алгоритмов модулярного умножения, реализованных программно. Измерение производительности для всех алгоритмов выполнялось над одним и тем же множеством (100 000) случайных полиномов. Тестирование проводилось на компьютере с процессором Intel® Celeron™ 800 MHz, ОЗУ 256 Mb, операционная система Microsoft Windows 2000 Professional. Программная реализация выполнена с помощью Microsoft Visual C++ 6. Тестирование проведено в полях, образованных неприводимыми полиномами $f_1(x) = x^{163} + x^7 + x^6 + x^3 + 1$, $f_2(x) = x^{233} + x^{74} + 1$.

В [4] проводилось аналогичное тестирование на рабочей станции с процессором Intel® Pentium II™ 400 MHz. Программная реализация алгоритмов выполнялась на языке программирования C. Результаты этого тестирования также приведены в таблице 5.

Таблица 5

Наименование	Наша реализация	Результаты из [4]
	m=233, мкс	m=233, мкс
Сложение	0,2	0,12
Алгоритм 2. Справа на лево, со сложением и сдвигом	18,86	27,14
Алгоритм 3. Справа на лево, со сложением и сдвигом, с шириной окна предвычислений q=4	12,556	–
Алгоритм 4. Справа на лево, комбинированный	6,52	12,01
Алгоритм 5. Слева на право, комбинированный	7,5	12,93
Алгоритм 6. Справа на лево, комбинированный метод с шириной окна предвычислений q=4	7,44	–
Алгоритм 7. Слева на право, комбинированный метод с шириной окна предвычислений q=4.	3,58	5,07
Алгоритм 8. Алгоритм Карацубы, рекурсивный	16,45	7,04
Алгоритм 9. Алгоритм Карацубы, рекурсивный, с предвычислениями	7,63	–
Алгоритм 9. Алгоритм Карацубы, итеративный, с предвычислениями	5,5	–

В таблице 6 приведем время выполнения операции приведение по модулю рассмотренными алгоритмами.

Таблица 6

Название алгоритма	Время работы, мкс.			
	Наша реализация		Результаты из [4]	
	m=163	*m=233*	*m=163*	*m=233*
Алгоритм 10. Приведение по модулю, побитного анализа делимого	66,0	96,6	–	–
Алгоритм 11. Приведение по модулю, побитного анализа делимого с предвычислениями	18,8	24,1	–	–
Алгоритм 12, 13. Приведение по фиксированному модулю	0,07	0,08	0,18	0,22

Отметим, что результаты проведенных тестов производительности зависят как от чистоты процессора, так и от размера кэш-памяти, к последнему показателю, особенно чувствительны алгоритмы с предвычислениями.

Выводы

Проведенные исследования позволяют нам сделать следующие выводы:

- наиболее эффективным алгоритмом «чистого» умножения оказался алгоритм 7, слева на право, комбинированный метод с шириной окна предвычислений $q = 4$;
- в тех случая, когда умножение используется для выполнения возведения в степень, алгоритмов с предвычислениями (6, 7), может быть повышена за счет однократного расчета таблицы предвычислений;
- дополнительным достоинством алгоритмов умножения 6 и 7, является возможность эффективного выполнения операции приведения по модулю «на лету», что позволяет сократить количество операций сложения машинных слов;
- так как в большинстве криптографических приложений в качестве образующих используются неприводимые полиномы специального вида (триномы и пентаномы), то операция приведения по модулю, может быть адаптирована под конкретный полином, что позволяет значительно повысить производительность, по сравнению с общим случаем;
- итеративный алгоритм Карацубы с предвычислениями, оказался недостаточно эффективным при программной реализации, это связано с необходимостью умножать полиномы степени которых меньше 32. Применение для этого наиболее эффективных (в

общем случае) алгоритмов 6 и 7 становится неоправданным. Кроме того, этот алгоритм не может быть совмещен с приведением по модулю «на лету». Однако эффективность алгоритма повышается в случае аппаратной поддержки операции умножения двух машинных слов.

Список литературы: 1. IEEE P1363 / D9 (Draft Version 9). Standard Specifications for Public Key Cryptography. 2. AMERICAN NATIONAL STANDARD X9.62–1998 (Draft version), Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). 3. *Hankreson D., Lopez Hernandez J., Menezes A.* Software implementation of elliptic curve cryptography over binary fields. Advances in Cryptology Crypto '99. 4. C.H. Lim, P.J. Lee, More flexible exponentiation with precomputation, In Advances in Cryptographi-CRYPTO'94, pp95-107, Springer-Verlag, 1994. 5. *J. Lopez, R. Dahab*, High-speed software multiplication in F_{2^m}. Technical report, IC-00-09, May 2000. 6. *Болотов А.А., Гашков С.Б., Фролов А.Б., Часовских А.А.*. Алгоритмические основы эллиптической криптографии: Учебное пособие. –М. 2000. – 100 с.