© Kovtun V., Bulakh M. Performance increasing methods for binary field inversion // Ukrainian Scientific Journal of Information Security, 2014, vol. 20, issue 1, p. 55-61.

МЕТОДЫ ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ОПЕРАЦИИ ИНВЕРТИРОВАНИЯ В ДВОИЧНОМ ПОЛЕ

Владислав Ковтун, Мария Булах

Национальный авиационный университет, Украина

КОВТУН Владислав Юрьевич, к.т.н.

Год и место рождения: 1978, Кировоград, Украина.
Образование: Харьковский Военный Университет, 2000.
Должность: доцент кафедры безопасности информационных технологий с 2010 года.
Научные интересы: информационная безопасность, быстрые арифметические преобразования в полях Галуа, криптосистемы с открытым ключом, криптоанализ криптографических преобразований с открытым ключом.
Публикации: более 50 научных публикаций, среди которых статьи в специализированых и зарубежных научных журналах, материалы конференций, патенты и др.
E-mail: vladislav.kovtun@gmail.com

БУЛАХ Мария Григорьевна

Год и место рождения: 1989, Львов, Украина.
Образование: Национальный авиационный университет, 2010.
Должность: соискатель кафедры безопасности информационных технологий с 2014 года.
Научные интересы: информационная безопасность, быстрые арифметические преобразования в полях Галуа, криптосистемы с открытым ключом, криптоанализ криптографических преобразований с открытым ключом.
E-mail: bulakh.masha@gmail.com

Аннотация. *Авторы предлагают ряд методов к увеличению производительности алгоритма мультипликативного инвертирования в двоичном поле на основе расширенного алгоритма Эвклида (РАЭ). Первый метод основывается на специфике самого РАЭ: инвариантный полином v, либо остается без изменений, либо обменивается содержимым с инвариантным полиномом u, это позволяет избежать необходимости вычисления степени полинома v. Второй метод основан на поиске «следующего подходящего индекса» при вычислении степени полинома, т.к. степень инвариантного полинома u уменьшается хотя бы на 1, то при дальнейшем вычисление степени полинома, можно учитывать текущее значение. Основываясь на втором методе, при модификации инвариантов, авторы предлагают использовать в вычислениях лишь значимые слагаемые, учитывая текущие степени пар полиномов (u,v) и (b,c). Предложенные методы позволяют увеличить производительность программной реализации инвертирования, для 32-х разрядных платформ, на 15-20%.*

Ключевые слова: *асимметрические криптографические преобразования, мультипликативное инвертирование, расширенный алгоритм Эвклида, двоичное поле, полином.*

Введение

Повсеместное использование информационно-телекоммуникационных систем в современном обществе, требует обеспечения защиты информации, которая циркулирует, создается, модифицируется, хранится и уничтожается. С этой целью, в каждой такой системе создается подсистема защиты информации, ядром которой является подсистема криптографической защиты информации. Среди криптографических преобразований, особое место занимают криптографические преобразования с открытым ключом, положенные в основу направленного шифрования, выработки общего секрета и электронной цифровой подписи.

Среди широко применяемых криптографических преобразований с открытым ключом выделяют: на эллиптических (ЭК) и гиперэллиптических кривых (ГЭК), в полях и кольцах. При реализации перечисленных криптопреобразований, активно используются операции над элементами поля $\mathbf{GF}(p)$ и $\mathbf{GF}(2^m)$. Известно, что поля вида $\mathbf{GF}(2^m)$ применяются при реализации криптографических преобразований на ЭК и полях, в международных стандартах IEEE P1363-2000 [1], ISO/IEC 15946-2 [2] и криптографических преобразований на ЭК в национальном стандарте ДСТУ 4145-2002 [3], ISO/IEC 15946-2 [2].

Согласно [1-4], криптографические преобразования с открытым ключом на ЭК можно представить в виде иерархии операций, Рис. 1, где особое место занимают операции в поле.

<table>
<tr><td>Криптопреобразования</td><td colspan="2">Зашифровывание/ расшифровывание</td><td colspan="2">Формирование и проверка цифровой подписи</td><td>Обмен ключами</td></tr>
<tr><td rowspan="2">Арифметика в группе точек эллиптической кривой</td><td colspan="5">Скалярное умножение точек эллиптической кривой</td></tr>
<tr><td colspan="3">Сложение точек</td><td colspan="2">Удвоение точки</td></tr>
<tr><td>Арифметика в поле целых чисел</td><td>Умножение</td><td>Сложение</td><td>Вычитание</td><td>Возведение в квадрат</td><td>Инвертирование</td></tr>
<tr><td>Команды CPU</td><td colspan="5">mov, mul, shr, shl, add, sub ...</td></tr>
</table>

Рис. 1. Иерархия операций в криптосистеме на ЭК

Известно [6-11], что производительность симметричных криптосистем существенно превосходит криптосистемы с открытым ключом, что позволяет говорить об актуальности исследований по увеличению производительности последних. Среди направлений дальнейших усовершенствований, авторы выделяют [6-11]:

− Повышение производительности операций в группе (точек ЭК, дивизоров в якобиане ГЭК и т.д.).

− Повышение производительности операций над структурами данных, используемых для представления элементов группы.

− Повышение производительности операций в поле.

− Повышение производительности операций над структурами данных, используемых для представления элементов поля.

− Оптимизация операций над структурами данных для современных суперскалярных процессоров [5].

Приоритетным направлением видится в повышения производительности операций над элементами поля. Далее, в работе, авторы ограничиваются полями вида $\mathbf{GF}\left(2^m\right)$.

На сегодняшний день существует значительное число публикаций по данному направлению [6-11], которые рассматривают алгоритмы операций над полиномами, даже в контексте архитектуры программной реализации таких библиотек [6-11], что позволяет существенно сократить накладные расходы на реализацию операций над полиномами в целом.

Среди операций над элементами поля $\mathbf{GF}\left(2^m\right)$ выделяют [4]: сложение, умножение, возведение в квадрат, мультипликативное инвертирование и возведение в степень. Исследования показывают [4, 7], что время выполнения операции инвертирования, существенно влияет на производительность криптографических преобразований на ЭК. Для уменьшения влияния операции инвертирования, предлагается [4] переходить к проективному представлению точек ЭК, аналогичные рассуждения применимы и для дивизоров якобиана ГЭК, однако полностью избавиться от инвертирования – невозможно.

В связи с этим, авторы концентрируют свое внимание на повышение производительности именно операции **инвертирования** в поле $\mathbf{GF}\left(2^m\right)$.

Описание алгоритма-прототипа расширенного алгоритма Эвклида и его модификация

Алгоритм мультипликативного инвертирования позволяет вычислять обратные не нулевые элементы $a \in GF\left(2^m\right)$, используя расширенный алгоритма Эвклида (РАЭ) для полиномов [4]. Алгоритм [4] основан на двух инвариантах $ba + df = u$ и $ca + ef = v$ для некоторых d и e, которые вычисляются неявно. В каждой итерации, если выполняется условие $\deg(u) \geq \deg(v)$, частичное распределение u через v, выполняя сдвиг $x^j v$ для u, где $j = \deg(u) - \deg(v)$. Следовательно, степень u останется постоянной, либо уменьшится на 1 и в среднем на 2. Складывая $x^j c$ с b позволяет сохранить инвариант. Алгоритм выполняет в среднем $\deg(a) = k$ итераций и останавливается, когда выполняется условие $\deg(u) = 0$, в этом случае $u = 1$ и $ba + df = 1$, следовательно $b = a^{-1} \bmod f(x)$.

Алгоритм 1. Расширенный алгоритм Эвклида для мультипликативного инвертирования в поле $\mathbf{GF}\left(2^m\right)$.

Вход: элемент $a \in GF\left(2^m\right)$, $a \neq 0$.

Выход: $a^{-1} \bmod f(x)$.

1. $b \leftarrow 1$, $c \leftarrow 0$, $u \leftarrow a$, $v \leftarrow f$.

2. While $\deg(u) \neq 0$ do

2.1. $j \leftarrow \deg(u) - \deg(v)$.

2.2. if $j < 0$ then $u \leftrightarrow v$, $b \leftrightarrow c$, $j \leftarrow -j$.

2.3. $u \leftarrow u + x^j v$, $b \leftarrow b + x^j c$.

3. Return (b).

Проведенный авторами анализ Алгоритма 1, позволяет выделить ряд аспектов для дальнейшего совершенствования РАЭ:

− На шаге п.2.2 и п.2.3 происходит модификация полинома u, в то время как полином v содержит предыдущее значение полинома u. Это позволяет отказаться от вычисления степени полинома v на шаге п.2.1. Что касается первичного

вычисления $\deg(v)$ на шаге п.2.1, то степень известна заведомо и является константой $\deg(v)=\deg(f)=m$.

– На шаге п.2.1 происходит вычисление степени полинома u. Исходя из общей логики алгоритма, степень полинома u постоянно уменьшается, хотя бы на 1. Это позволяет отказаться от вычисления степени $\deg(u)$, на каждой итерации в общем виде, а лишь заниматься ее уточнением основываясь на текущем значении степени.

– На шаге п.2.3 производится сдвиг полинома v и последующее, его сложение с u, однако следует заметить, что в процессе выполнения цикла п.2 степень полиномов v и u постоянно уменьшается, а степень полиномов b и c постоянно растет. Это позволяет выполнять сдвиг и сложение не всех элементов массива, в котором представлены элементы поля, а лишь значимые – заведомо отличные от нуля.

Произведем оценку сложности Алгоритма 1, в среднем. Пусть степень полинома $\deg(a)=k$ и вес по Хеммингу полинома $h=\text{weight}(a)=k/2$. Тогда количество итераций цикла п.2 составит k, и число истинных условий $j\neq 0$, составит $2k/3$, в оставшихся $k/3$ случаях сдвиг не выполняется. Сложность Алгоритма 1, в среднем, составит:

$$I_{avr}(A_1)=k\left(2I_{\deg}+2I_{add}+2I_{shl}\right)+2k/3\left(2I_{swp}\right), \qquad (1)$$

где $I_{\deg}$ - сложность алгоритма вычисления степени полинома, I_{add} - сложность алгоритма сложения двух полиномов, I_{shl} - сложность алгоритма сдвига на произвольное число бит (может превышать длину машинного слова), I_{swp} - сложность алгоритма обмена двух полиномов. В дальнейшем, сложностью I_{swp} можно пренебречь в силу использования указателей, что не потребует выполнения значительного числа переприсвоений.

В упрощенном виде (1) можно будет представить:

$$I_{avr}(A_1)=k\left(2I_{\deg}+2I_{add}+2I_{shl}\right). \qquad (2)$$

Несложно заметить из (2), что уменьшение сложности алгоритма в целом, может быть достигнуто за счет уменьшения числа операций и сложности $I_{\deg}$, I_{add} и I_{shl}.

Детального рассмотрения потребует непосредственно сам алгоритм вычисления степени произвольного полинома a. Заведомо предполагается, что для представления элементов поля $\mathbf{GF}(2^m)$, используется полиномиальный базис.

Элемент поля $b\in\mathbf{GF}(2^m)$ в полиномиальном базисе представляется двоичным вектором $b_{m-1}x^{m-1}+b_{m-2}x^{m-2}+\cdots+b_1x^1+x_0$, таким, что можно представить в виде $a_{n-1}^{(w)}2^{(n-1)w}+a_{n-2}^{(w)}2^{(n-2)w}+\cdots+a_1^{(w)}2^w+a_0^{(w)}$ массива машинных слов с двоичной длиной w, где $n=\lceil\frac{m}{w}\rceil$ - число машинных слов необходимых для представления полином двоичной длины m; b_i - двоичные коэффициенты; $a_j^{(w)}$ - машинные слова двоичной длины w.

Идея алгоритма вычисления степени полинома состоит в поиске номера самого старшего элемента массива $a_j^{(w)}$, отличного от нуля, и нахождения номера самого старшего единичного бита в найденном элементе массива $a_j^{(w)}$. Поиск самого старшего элемента массива $a_j^{(w)}$ заключается в последовательном переборе элементов массива, начиная с конца, до тех пор, пока не встретится искомый. Что касается определения номера старшего единичного бита в элементе массива (машинном слове), то известно достаточно много алгоритмов [12], которые требуют опять же последовательного перебора, т.е. значительных вычислительных ресурсов. Повысить производительности вычисления номера старшего единичного бита в слове, авторами, предлагается воспользоваться целым рядом известных «трюков» [12], основанных на битовых операциях машинных слов. Наработки предложенные авторами представлены в виде Алгоритма 2. Проведем его более детальное рассмотрение. На шаге п.2, в цикле ищется самый старший элемента массива – машинное слово, отличное от нуля (содержащего хотя бы один отличный от нуля бит). Поиск производиться с самого старшего машинного слова, к младшему. На шаге п.3, фиксируется отличный нуля элемент массива и его номер. Номер элемента, в дальнейшем, будет использовано для вычисления степени полинома.

Алгоритм 2. Алгоритм вычисления степени полинома в поле $\mathbf{GF}(2^m)$.

Вход: $a\in\mathbf{GF}(2^m)$; $n=\lceil\frac{m}{w}\rceil$, где n -число машинных слов, которое занимает полином; w - ширина машинного слова, обычно $w=32$.

Выход: $\deg(a)$.

1. $i\leftarrow n-1$.

2. While $\left(a_i^{(32)}\neq 0\right)\&\&(i>0)$

2.1. $i\leftarrow i-1$.

3. $t\leftarrow a_i^{(32)}$.

4. $t\leftarrow t\,|\,(t>>1)$, $t\leftarrow t\,|\,(t>>2)$, $t\leftarrow t\,|\,(t>>4)$, $t\leftarrow t\,|\,(t>>8)$, $t\leftarrow t\,|\,(t>>16)$.

5. $t\leftarrow t-\left((t>>1)\,\&\,0\text{x}55555555\right)$,

$t\leftarrow\left(t\,\&\,0\text{x}33333333\right)+\left((t>>2)\,\&\,0\text{x}33333333\right)$,

$t\leftarrow\left(\left(t+(t>>4)\,\&\,0\text{xf0f0f0f}\right)\cdot 0\text{x}1010101\right)>>24$.

6. Return $\left((i<<5)+t-1\right)$.

На шаге п.4 применяется «трюк» [12] - формирование «маски», который позволяет заполнить единичными битами все младшие биты от самого старшего –. Далее необходимо подсчитать

количество единичных битв в машинном слове, чтобы определить номер старшего бита.

Подсчет всех единичных битов в машинном слове, на шаге п.5, выполняется «трюк» [12], причем их число будет на единицу больше чем, номер самого старшего отличного от нуля бита.

Степень полинома, вычисляется на шаге п.6, где учитывается общее число бит в каждом машинном слове, номер самого старшего отличного от нуля машинного слова, а также номер старшего бита в старшем слове.

Описанные ранее результаты анализа Алгоритма 1, позволяют внести следующие модификации:

− На каждой итерации цикла п.2, полином v является постоянным, либо может быть обменян с u, т.е. можно не вычислять его степень на каждой итерации цикла: степень полинома v остается либо постоянной, либо равна степени полинома u. Так можно сэкономить на одной операции вычисления степени полинома, на каждой итерации цикла п.2. Количество итераций цикла п.2. сопоставимо со степенью полинома $a \in \mathbf{GF}(2^m)$, который полежит инвертированию.

− На шаге п.2.1 производится вычисление степени полинома u, степень которого на шаге п.2.3 уменьшается, хотя бы на 1, т.е. можно не вычислять его степень на каждой итерации цикла, а лишь уточнять: степень полинома u уменьшается хотя бы на 1, что позволяет воспользоваться методом «следующего подходящего», для фиксации позиции текущего машинного слова. Такой подход позволяет начинать производить поиск не с самого старшего машинного слова, а с того, на котором остановился на предыдущей итерации цикл п.2. Отметим, что количество проверок при поиске отличного от нуля машинного слова в цикле п.2 уменьшилось в 2 раза.

− На шаге п.2.3 производится сдвиг полинома v и последующее, его сложение с полиномом u, степень которых непрерывно уменьшается на протяжении всего цикла п.2, а степень полиномов b и c постоянно растет. Здесь можно воспользоваться методом «следующего подходящего», для оперирования лишь со значимыми элементами массива машинных слов. Это позволяет выполнять сдвиг и сложение не всех элементов массива, в котором представлены элементы поля, а лишь значимые. Отметим, что количество сложений и сдвигов машинных слов в цикле п.2 уменьшилось почти в 2 раза.

Учтя в Алгоритме 1, результаты анализа, проведенного авторами, а также сам алгоритм вычисления степени полинома Алгоритм 2, был предложен модифицированный РАЭ (МРАЭ) для мультипликативного инвертирования в поле $\mathbf{GF}(2^m)$, который представлен в виде Алгоритма 3.

Алгоритм 3. Модифицированный расширенный алгоритм Эвклида для мультипликативного инвертирования в поле $\mathbf{GF}(2^m)$.

Вход: $a \in \mathbf{GF}(2^m)$, $a \neq 0$, $n = \lceil \frac{m}{w} \rceil$, где n -число машинных слов, которое занимает полином; w - ширина машинного слова, обычно $w = 32$.

Выход: $a^{-1} \bmod f(x)$.

1. $b \leftarrow 1$, $c \leftarrow 0$, $u \leftarrow a$, $v \leftarrow f$.

2. $i \leftarrow n-1$.

3. While $\left(u_i^{(32)} \neq 0\right) \& \& (i > 0)$

3.1. $i \leftarrow i-1$.

4. $t \leftarrow u_i^{(32)}$.

5. $t \leftarrow t \,|\, (t >> 1)$, $t \leftarrow t \,|\, (t >> 2)$, $t \leftarrow t \,|\, (t >> 4)$, $t \leftarrow t \,|\, (t >> 8)$, $t \leftarrow t \,|\, (t >> 16)$.

6. $t \leftarrow t - \left((t >> 1) \,\&\, 0\text{x}55555555\right)$,
$t \leftarrow (t \,\&\, 0\text{x}33333333) + \left((t >> 2) \,\&\, 0\text{x}33333333\right)$,
$t \leftarrow \left(\left(t + (t >> 4) \,\&\, 0\text{xf0f0f0f}\right) \cdot 0\text{x}1010101\right) >> 24$.

7. $\deg U = \left((i << 5) + t - 1\right)$.

8. $\deg V = m$.

9. While $(\deg U > 0)$ do

9.1. If $(\deg U < \deg V)$ then $k \leftarrow \deg V - \deg U$, $u \leftrightarrow v$. $b \leftrightarrow c$.

9.2. else $k \leftarrow \deg U - \deg V$.

9.3. if $(k > 0)$ then $u = u + x^k \cdot v$, $b = b + x^k \cdot c$.

9.4. else $u = u + v$, $b = b + c$.

9.5. If $(\deg U < \deg V)$ then $\deg V = \deg U$.

9.6. While $\left(u_i^{(32)} \neq 0\right) \& \& (i > 0)$

9.6.1. $i \leftarrow i-1$.

9.7. $t \leftarrow u_i^{(32)}$.

9.8. $t \leftarrow t \,|\, (t >> 1)$, $t \leftarrow t \,|\, (t >> 2)$, $t \leftarrow t \,|\, (t >> 4)$, $t \leftarrow t \,|\, (t >> 8)$, $t \leftarrow t \,|\, (t >> 16)$.

9.9. $t \leftarrow t - \left((t >> 1) \,\&\, 0\text{x}55555555\right)$,
$t \leftarrow (t \,\&\, 0\text{x}33333333) + \left((t >> 2) \,\&\, 0\text{x}33333333\right)$,
$t \leftarrow \left(\left(t + (t >> 4) \,\&\, 0\text{xf0f0f0f}\right) \cdot 0\text{x}1010101\right) >> 24$.

9.10. $\deg U = \left((i << 5) + t - 1\right)$.

10. Return (b).

На шаге п.1 производится инициализация полиномов b, c, u и v, которые будут модифицироваться на протяжении работы алгоритма. Далее на шагах п.2-3 определяется индекс i старшего, отличного от нуля, машинного слова, полинома u. Поиск начинается со старших машинных слов.

Найденное машинное слово $u_i^{(32)}$, используется на шагах пп.4-6 для вычисления индекса старшего бита t в слове $u_i^{(32)}$, и на шаге п.7 вычисляется степень полинома u. Начальная степень полинома v, известна заранее и равна m. Для вычисления индекса старшего бита используется «трюк» [12] для выделения старшего бита – все младшие бит устанавливаются в единицу

и затем производится подсчет числа единиц в слове, с помощью другого «трюка» [12].

Цикл п.9 является основным для всего алгоритма и выполняется, до тех пор, пока степень полинома u, отлична от 0, в среднем число итераций $\deg(a)=k$. В цикле, на шаге п.9.1 производится проверка степени $\deg U$ и $\deg V$ полиномов u и v, соответственно. В случае $(\deg U<\deg V)$, необходимо произвести обмен содержимого полиномов u, v и b, c, соответственно. В среднем, вероятность случая $(\deg U<\deg V)$ составляет $k/3$. Вычисляется разница $(\deg U-\deg V)$, которая в дальнейшем используется для сдвига полиномов v и c. На шагах п.9.3 и п.9.4 производится очистка старшего бита полинома u и установка соответствующих битов полинома b. Отметим, что на шагах п.9.3 и п.9.4 следует оперировать лишь значимыми машинными словами, т.е. теми которые содержат биты меньшие $\deg(b)$ и $\deg(v)$. Учитывая, что степень полинома b растет, а степень u - уменьшается, причем $\deg(b)+\deg(u)=m$.

Вычисление степени полинома v осуществляется на шаге п.9.5, причем степень полинома меняется если выполнилось условие $(\deg U<\deg V)$ обмена полиномов u и v. Далее на шаге п.9.6 и п.9.7 определяется индекс i старшего, отличного от нуля, машинного слова, полинома u. Найденное машинное слово $u_i^{(32)}$, используется на шагах пп.9.8-9.9 для вычисления индекса старшего бита t в слове $u_i^{(32)}$, и на шаге п.7 вычисляется степень $\deg U$ полинома u.

По завершению цикла п.9, алгоритм возвращает полином b, такой что $b=a^{-1}\bmod f(x)$.

Произведем оценку сложности Алгоритма 3, в среднем. Пусть степень полинома $\deg(a)=k$ и вес по Хеммингу полинома $h=\mathrm{weight}(a)=m/2$. Тогда количество итераций цикла п.2 составит m, и число истинных условий $j\neq 0$, составит $2k/3$. Сложность Алгоритма 3, в среднем, составит:

$$I_{avr}(A_3)=k\left(I_{\deg}+\tfrac{1}{2}\left(2I_{add}+2I_{shl}\right)\right)+2k/3\left(2I_{swp}\right), \qquad (3)$$

В дальнейшем, сложностью I_{swp} можно пренебречь в силу использования указателей, что не потребует выполнения значительного числа переприсвоений. В упрощенном виде (3) можно будет представить:

$$I_{avr}(A_3)=k\left(I_{\deg}+I_{add}+I_{shl}\right). \qquad (4)$$

Особенности программной реализации

Для подтверждения теоретических оценок эффективности на практике МРАЭ, в соответствии с предложенными методами, необходимо программно реализовать известный РАЭ и предложенный МРАЭ. Полученные программные реализации экспериментально исследовать для оценки среднего времени выполнения операции на полиномах различной длины [13].

При программной реализации, авторами использовалось полиномиальное представление полиномов поля $\mathbf{GF}(2^m)$, которые в свою очередь представлялись в виде массивов фиксированной длины для заданного m.

При написании программ, учитывалась особенность суперскалярной архитектуры 32-х разрядных процессоров и возможностями современных компиляторов по предсказанию переходов, параллельному выполнению команд, разворачиванию циклов и т.д., в соответствии с [5].

Программная реализация выполнялась на языке высокого уровня C++, в среде Microsoft Visual Studio, в конфигурации Release с помощью компилятора Microsoft Visual C++ 2010 (/O3, поддержка SSE2) и Intel C++ Compiler XE2013 (/O3, поддержка SSE4.2) для 32-разрядных платформ. В дальнейшем будем пользоваться сокращенными названиями MCC и ICC, соответственно.

Сравнение известного и предложенного алгоритмов

Для оценки эффективности предложенных методов по модификации алгоритма мультипликативного инвертирования в поле $\mathbf{GF}(2^m)$, авторами выполнена программная реализация Алгоритмов 1 и 3, в соответствии с условиями, приведенными в предыдущем разделе. В программной реализации усредняется время выполнения 1 млн. операций, которое приведено в таблице 1. В экспериментах было учтено, что степень инвертируемого полинома, может влиять на количество итераций основного цикла, поэтому рассматривались полиномы степени близкой к максимальной, для полей, которые используются в криптографических преобразованиях из ДСТУ 4145-2002 [3] и FIPS-186-3 [13].

Эксперименты производились для различных степеней расширений двоичных полей m [3, 13] на наиболее распространенных мобильных процессорах Intel Core i3 M350 и настольных процессорах 3-го поколения Intel Core i5-3570 и 4-го поколения Intel Core i5-4670 под управлением ОС Windows 7 SP 1 x86-64.

Замеры времени выполнения операции инвертирования приведены в таблице 1.

Результаты экспериментов показывают, что степень инвертируемого полинома не влияет на время вычисления обратного элемента с помощью РАЭ, в то время как МРАЭ показывает линейное уменьшение времени вычисления обратного элемента с уменьшением его степени. Результаты экспериментов для элементов поля $\mathbf{GF}(2^m)$ со степенями меньшими m в работе не приводятся.

Таблица 1

Результаты экспериментальной оценки времени выполнения операции инвертирования

#	Время, мкс											
	Intel Core i3-350M				Intel Core i5-3570				Intel Core i5-4670			
	ICC XE2013		MCC2010		ICC XE2013		MCC2010		ICC XE2013		MCC2010	
	Inv	Inv*	Inv	Inv*	Inv	Inv*	Inv	Inv*	Inv	Inv*	Inv	Inv*
Степень полинома близкая к m												
89	6,71	5,44	6,27	5,10	2,53	1,95	2,76	1,84	2,53	1,95	2,37	1,84
163	16,08	11,34	14,38	11,96	6,85	4,05	6,33	4,65	6,85	3,95	6,13	4,05
191	18,17	14,52	17,38	14,71	7,73	5,46	7,85	5,48	7,73	5,26	7,65	5,48
233	27,13	22,12	24,57	19,39	11,80	7,04	11,59	7,65	11,80	7,02	11,02	6,90
257	31,56	25,18	27,93	24,54	13,21	7,99	13,33	8,56	12,17	7,81	12,33	7,60
307	37,72	30,45	34,24	26,11	17,98	11,11	17,64	12,41	17,68	9,58	17,54	11,41
367	53,18	42,17	46,05	33,81	23,35	14,78	21,84	16,63	22,35	13,18	21,64	14,63
409	61,31	40,18	54,48	47,76	26,41	16,97	26,81	18,51	25,97	14,93	26,41	17,51
431	67,75	54,24	59,10	44,03	28,99	17,86	29,29	19,25	28,27	17,00	28,99	18,25
571	103,44	64,86	94,42	70,26	46,46	25,47	44,87	26,98	43,39	24,64	44,67	26,83

* - модифицированный алгоритм, с предложенными методами оптимизации.

Для процессора Intel Core i3-350M, МРАЭ показал выигрыш в 18-21% (ICC) и 16-18% (MCC), в тоже время MCC реализация оказалась эффективнее на 2,2% чем на ICC. Для процессора Intel Core i5-3570, МРАЭ показал выигрыш в 19-22% (ICC) и 17-22% (MCC), в тоже время ICC реализация оказалась эффективнее на 4,9% чем на MCC. Для процессора Intel Core i5-4670, МРАЭ показал выигрыш в 18-25% (ICC) и 14-22% (MCC), в тоже время ICC реализация оказалась эффективнее до 14,7% чем MCC.

Полученные временные оценки полностью соответствуют теоретическим оценкам вычислительной сложности в (2) и (4). Причем программная реализация предложенного МРАЭ показала лучшую производительность на 20-50% (для мобильного процессора Intel Core i3-350M с меньшей тактовой частотой и настольным процессором Intel Pentium 4 530J с вдвое большей тактовой частотой), чем в работе [7], для соответствующих полей.

Применении МРАЭ в программной реализации алгоритмов формирования и проверки ЭЦП в соответствии с ДСТУ 4145-2002 [3], в которой использовался алгоритм скалярного умножения точек «справа-налево» [4]:

− в аффинном представлении точек позволило увеличить производительность на 16-20%, с увеличением размера поля;

− в проективном представлении точек Лопеса-Дахаба [4] позволило увеличить производительность на 2-4%, с увеличением размера поля.

Приведенные сравнения применимы к результатам полученным с помощью Intel C++ Compiler XE2013 и Visual C++ 2010 для архитектуры x86.

Заключение

По результатам проведенных исследований можно сделать **следующие выводы:**

1. Предложенные авторами методы оптимизации алгоритма мультипликативного инвертирования в поле $\mathbf{GF}(2^m)$, позволили уменьшить вычислительную сложность МРАЭ в 2 раза, что подтверждается экспериментальными оценками.

2. Программная реализация МРАЭ, обладает, в среднем, большей производительностью на 15-20%, чем алгоритма прототипа.

3. Применение модифицированного алгоритма инвертирования, в программной реализации алгоритмов формирования и проверки ЭЦП на основе ДСТУ 4145-2002, позволило увеличить производительность на 16-20% в аффинном представлении точек и 2-4% в проективном представлении Лопеса-Дахаба, соответственно.

4. Предложенные реализации МРАЭ инвертирования не ориентированы на многопоточное выполнение, что не позволило полностью реализовать потенциал современных многоядерных процессоров.

5. Развитие современных микропроцессоров направлено на увеличение числа потоков выполнения команд, что в свою очередь требует разрабатывать полноценные алгоритмы для эффективной программной реализации на перспективных микропроцессорах. Так, компания NVIDIA, уже сейчас предлагает графические процессоры с числом ядер более 512, а также удобную среду разработки CUDA [14-18], для создания полноценных многопоточных приложений. Дальнейшее направление исследований будет направлено на изучение и эффективное распараллеливание алгоритмов арифметических операций в поле $\mathbf{GF}(2^m)$.

Литература

[1] IEEE Std 1363-2000: Standard Specifications For Public Key Cryptography. Institute of Electrical and Electronics Engineers, 2000. — 228 p. URL: http://grouper.ieee.org/groups/1363/

[2] ISO/IEC 15946-2:2002. Information technology — Security techniques — Cryptographic techniques based on elliptic curves —Part 2: Digital signatures, 2007, 36 p.

[3] ДСТУ 4145-2002. Информационные технологии. Криптографическая защита информации. Цифровая подпись, основанная на эллиптических

кривых. Формирование и проверка. — К. : Держстандарт України, 2003. — 39 с.

[4] D. Hankerson, J. Hernandez, A. Menezes. Software implementation of Elliptic Curve Cryptography over binary fields. Proceedings of Workshop on Cryptographic Hardware and Embedded System, 2000, LNCS Vol. 1965, pp. 1-24.

[5] Intel® 64 and IA-32 Architectures Optimization Reference Manual. Order Number: 248966-025, 2013, 660 p. URL: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html

[6] Giorgi P. Izard T, Tisserand A. Comparison of Modular Arithmetic Algorithms on GPUs // ParCo'09: International Conference on Parallel Computing, France, 2009, 8 p. URL: http://hal-lirmm.ccsd.cnrs.fr/lirmm-00424288/fr/

[7] Zhijie Jerry Shi and Hai Yan. Software Implementations of Elliptic Curve Cryptography // International Journal of Network Security, 2008, Vol.7, No.2, pp. 57–166.

[8] TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. URL: http://discovery.csc.ncsu.edu/software/TinyECC/

[9] Galois Field Arithmetic Library. URL: http://www.partow.net/projects/galois/

[10] MPFQ: Fast Finite Fields Library. URL: http://mpfq.gforge.inria.fr/

[11] LibTom Projects: LibTomPoly. URL: http://libtom.org

[12] Sean Eron Anderson. Bit Twiddling Hacks. URL: http://graphics.stanford.edu/~seander/bithacks.html

[13] National Institute of Standards and Technology, Recommended Elliptic Curves for Federal Government Use, Appendix to FIPS 186-2, 2000, 43 p.

[14] NVIDIA. NVIDIA CUDA Programming Guide 5.0, 2013, 214 p. URL: http://docs.nvidia.com/cuda/cuda-c-programming-guide/

[15] M. Bluhm, S. Gueron. Fast Software Implementation of Binary Elliptic Curve Cryptography // Cryptology ePrint Archive, Report 2013/741, 2013, 19 p. URL: http://eprint.iacr.org/2013/741.pdf

[16] E. M. Mahé, J.-M. Chauvet. Fast GPGPU-Based Elliptic Curve Scalar Multiplication // Cryptology ePrint Archive, Report 2014/198, 2014, 9 p. URL: http://eprint.iacr.org/2014/198.pdf

[17] Aaron E. Cohen, Keshab K. Parhi. GPU accelerated elliptic curve cryptography in GF(2^m) // 53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2010, pp. 57-60.

[18] Tony Cheneau, Aymen Boudguiga, Maryline Laurent. Significantly Improved Performances Of The Cryptographically Generated Addresses Thanks To ECC And GPGPU // Computers and Security Journal (CoSe), Elsevier, 2010, Vol. 29, pp. 419-431.

УДК 004.051/056 (045)

Ковтун В.Ю., Булах М.Г. Методи підвищення продуктивності операції інвертування у двійковому полі

Анотація. *Автори пропонують ряд методів збільшення продуктивності алгоритму мультиплікативного інвертування у двійковому полі на основі розширеного алгоритму Евкліда. Перший метод ґрунтується на специфіці самого розширеного алгоритму Евкліда: інваріантний поліном v або залишається без змін, або обмінюється вмістом з інваріантним поліномом u, це дозволяє уникнути необхідності обчислення ступеня полінома. Наступний метод заснований на методі «наступного підходящого індексу» при обчисленні ступеня полінома: враховуючи той факт, що в процесі роботи розширеного алгоритму Евкліда, ступінь інваріантного полінома u зменшується хоча б на 1, то при подальшому обчисленні ступеня полінома можна враховувати поточне значення ступеня. Запропоновані методи дозволяють збільшити продуктивність програмної реалізації інвертування для 32-х розрядних платформ на 15-20%.*

Ключові слова: *асиметричні криптографічні перетворення, мультиплікативне інвертування, розширений алгоритм Евкліда, двійкове поле, поліном.*

Kovtun V., Bulakh M. Performance increasing methods for binary field inversion

Abstract. *Authors propose several methods for increasing performance of multiplicative inversion algorithm in binary fields based on Extended Euclidean Algorithm. First method is based on Extended Euclidean Algorithm specific: either invariant polynomial v is a same or swaps with invariant polynomial u. Thus, it does not necessary to compute degree of polynomial v. Next method is based on «next fit element» in polynomial degree computation: on each iteration of Extended Euclidean Algorithm execution, degree of invariant polynomial u decreases at list one. On next iteration Extended Euclidean Algorithm degree searches from where it left off the previous time. Proposed methods allow to increase performance of software implementation of inversion in binary field for 32-bit architectures on 15-20%.*

Key words: *asymmetric cryptography transformation, multiplicative inversion, Extended Euclidean Algorithm, binary field, polynomial.*

Отримано *13 лютого 2014 року*, затверджено редколегією *5 березня 2014 року*