В.Ю.Ковтун, А.А.Охрименко

МЕТОД ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ОПЕРАЦИИ ПРИВЕДЕНИЯ ПО ПРОСТОМУ МОДУЛЮ

Аннотация. *Авторы предлагают подход к увеличению производительности алгоритма приведения по простому модулю на основе классического алгоритма приведения по модулю Барретта для 32-х разрядных платформ. Предлагается рассматривать алгоритма Барретта, как совокупность алгоритмов Комба для умножения целых чисел с отложенным переносом, в которых учитываются следующие факторы: различная длина множителей, у которых используются лишь старшие или младшие части произведений. Использование частичных умножений чисел в алгоритме Комба, позволяет избежать избыточных операций при самом умножении алгоритмом Комба, а также операций сдвига произведений, после частичного умножения.*
Ключевые слова: *целые числа, простые числа, умножение, частичное умножение, приведение по модулю Барретта.*

Abstract*. The authors propose an approach to increase the performance of the modular reduction algorithm based on classical algorithm of Barrett modular reduction for 32-bit platforms. Barrett's algorithm is proposed to consider, as a set of Comba integer multiplication algorithms with delayed carry, and take into account the following factors: different length of multipliers, which are used only high or low parts of the products . Using partial integer multiplications in Comba algorithm avoids the redundant operations of multiplication and products shift operations after partial multiplication.*
Keywords: *integers, prime numbers, multiplication, partial multiplication, Barrett modular reduction.*

Введение

Современные криптографические преобразования с открытым ключом получили широкое распространение в средствах защиты информации в различных информационно-телекоммуникационных системах. В основе большинства из них лежат трудноразрешимые задачи факторизации большого числа, задачи дискретного логарифма в поле целых чисел, а также основывающиеся на решении задачи дискретного логарифма в группе точек эллиптической кривой или якобиане дивизоров гиперэллиптических кривых. Несмотря на существенные отличия криптографических преобразований в трудноразрешимых задачах, все они используют арифметику в поле GF(p)

целых чисел. Известно, что поле $GF(p)$ является кольцом $Z_p = \{0, 1, \ldots, p-1\}$ вычетов по модулю простого числа p [3]. Отсюда, выполнение любой арифметической операции в поле, будь то умножение, сложение, вычитание, возведение в квадрат или произвольную степень, вычисление квадратного корня, требует выполнения условия замкнутости указанных операций. Исходя из факта, что производительность криптосистем (ассиметричных) с открытым ключом существенно уступает симметричным криптосистемам, можно говорить об актуальности дальнейшего повышения их производительности. Возможным направлением повышения производительности криптосистемы в целом, является повышение производительности операций над целыми числами. Среди наиболее часто используемых операций в поле, можно выделить операцию умножения и сложения по модулю простого числа, в которых явным образом используется операция приведения по модулю. Таким образом, можно сделать вывод, что производительность операции приведения по модулю оказывает сильное влияние на производительности криптосистемы в целом, что позволяет говорить в данной работе о необходимости дальнейшего увеличения производительности именно операции приведения по модулю простого числа.

Известно большое число публикаций по данному направлению [1, 4], которые рассматривают не только сами алгоритмы операций над целыми числами, но и предлагают различные программные архитектуры библиотек для работы с целыми числами криптографической длины, позволяющих существенно сократить накладные расходы на реализацию операций над числами в целом.

На рис. 1 приводится классификация алгоритмов приведения по модулю, учитывающая различные аспекты самих, алгоритмов, направленных на их оптимизацию. Авторами выделяется:

1. Представление модуля – целого числа. Учитывая специфику представления (кодирования) целых чисел, можно существенно уменьшить число элементарных операций (процессорных инструкций). Выделяют следующие представления:

- Двоичное (обычное).
- Четверичное.
- Двоичное сокращенное (двоичное DRF).
- Избыточное статическое и избыточное динамическое.
- Несмежное (NAF).
- Остаточных классов (RNS).

Рис. 1. Классификация алгоритмов приведения по модулю

2. Простой модуль. Учитывая специфику представления модуля в двоичном виде, можно оптимизировать алгоритм приведения. Выделяют следующие классы простых чисел:

- Простые числа общего вида. Представляют собой универсальные алгоритмы, основанные на делении «в столбик», Барретта и Монтгомери.
- Простые числа, являющиеся обобщенными мерсеновыми и псевдо-мерсеновыми. Представляют собой специализированные алгоритмы, оптимизированные под обобщенными мерсеновые, так и конкретные простые числа [2].

3. Порядок анализа. Учитывая специфику направления анализа, с начала или конца чисел, выделяют:

- С начала (младшие биты) в алгоритме Монтгомери [5].
- С конца (старшие биты) в алгоритме Барретта [1].
- Двусторонние и многосторонние.

4. Алгоритм. Учитывая различные подходы к приведению, выделяют несколько основных алгоритмов и их модификаций [3]:

- Классический («школьный», «деление в столбик»).
- Монтгомери [4] и его модификации [2].
- Барретта [1] и его модификации.
- Универсальный и жестко запрограммированный специальный модуль.

5. Приближение. Учитывая тот, факт, что не во всех случаях имеет смысл получать полностью приведенное число, достаточно его привести к виду, незначительно превышающего модуль. Данный подход может быть применен, ко всем известным на сегодня алгоритмам. Можно выделить несколько основных алгоритмов:

- Частичное приведение для алгоритма Монтгомери.
- Частичное приведение для алгоритма Барретта.
- Частичное приведение для классического алгоритма.
- Частичное приведение для модуля специального вида.

6. Распараллеливание. Учитывая, тот факт, что большинство современных компьютеров, обладают несколькими процессорами или процессором с несколькими ядрами, появилась возможность распараллелить алгоритмы приведения по модулю. Можно выделить несколько основных алгоритмов:

- Двустороннее частичное приведение для алгоритма Монтгомери.
- Многостороннее частичное приведение для алгоритма Монтгомери.
- Двустороннее частичное приведение для алгоритма Барретта.

- Многостороннее частичное приведение для алгоритма Барретта.

7. Предвычисления. Учитывая тот факт, что при выполнении операции приведения по модулю, используется одно и то же простое число – модуль, то возникает возможность некоторые промежуточные значения предвычислить заранее. Данный подход может быть применен, ко всем известным на сегодня алгоритмам. Механизм получил развитие в следующих алгоритмах:

- Однократное перекрытие алгоритма Барретта.
- Итеративное (чередующееся) перекрытие алгоритма Барретта [2].
- Использование одного и нескольких предвычисленных значений.
- Таблица предвычислений для простого общего вида (большая).
- Таблица предвычислений для простого специального вида DRF (большая).
- Таблица предвычислений для модуля специального вида DRF (маленькая).

8. Без предвычислений констант. Учитывая тот факт, что в алгоритмах Монтгомери и Барретта, необходимо предварительное вычисление ряда констант. Известен ряд работ, которые позволяют уйти от предварительного вычисления констант:

- Без вычислений констант для алгоритма Монтгомери (алгоритм масштабирования Куискотера) [2].
- Без вычислений констант для алгоритма Барретта (алгоритм масштабирования Куискотера) [2].

9. Отложенные операции. Учитывая, тот факт, что при выполнении арифметических операций, часто операции умножения чередуются со сложением, вычитанием, получил развитие подход, в котором приведение по модулю выполняется лишь, в случае достижения результата арифметических операций некоторой границы. Однако такой подход требует постоянного контроля над достижением границы, требует оперировать операндами переменной длины, что приводит к увеличению числа элементарных операций.

10. Место применения. Следует выделить подход, при котором производиться приведение по модулю [2]:

- После выполнения операции (умножение, возведение в квадрат, сдвиг влево, сложение, вычитание) (данная работа).
- В процессе выполнения операции (умножение, возведение в квадрат, сдвиг влево).

11. Алгоритм умножения. Следует заметить, что в алгоритмах Барретта и Монтгомери используются операции умножения целых чисел различной длины, от реализации которых напрямую зависит производительность самого приведения. Известно применение умножения по:

- Алгоритму Комба (данная работа).
- Алгоритму Карацубы.

За основу было выбрано направление, в котором отделяется алгоритм основной операции от алгоритма приведения по модулю, например, сначала выполняется умножение, а лишь затем производиться приведение по модулю.

Интерес представляет подход «ленивого» или отложенного приведения (Lazy/Delayed Reduction) предложенный в работах, который позволяет выполнять приведение по модулю большого числа лишь, когда выполнится одно из условий:

- После арифметических операцией, когда полученный результат достиг границы допустимой длины.
- После арифметических операцией, когда полученный результат, необходимо интерпретировать как элемент поля (меньшего модуля).
- Перед арифметическими операциями, когда полученный результат может превысить границу допустимой длины.

Однако применение «ленивого» приведения выходит за рамки данной работы и будет рассмотрен в дальнейших исследованиях.

Сформулируем уровни оптимизации программной реализации алгоритма приведения по модулю, которые могут быть применены, Рис. 2:

- Инструкции процессора. Предполагается, что современные компиляторы могут наилучшим образом оптимизировать исходный код приложения на высокоуровневом языке для выбранной процессорной архитектуры.
- Представление больших целых чисел. Наиболее распространенным является двоичное представление (система исчислений), однако существуют и другие преставления (системы исчислений), которые позволяют существенно повысить производительность операций над целыми числами (рассматривались ранее).
- Вид больших простых чисел. Учитывая, что в зависимости от вида простого числа общего вида, обобщенного Мерсенового, псевдо-Мерсенового числа, можно оптимизировать алгоритм приведения непосредственно под конкретное простое число.
- Представление структур, данных для больших целых чисел.

Рис. 2. Уровни оптимизации алгоритмов приведения по модулю

В работе будет рассматриваться подход к дальнейшему совершенствованию алгоритма приведения по модулю Барретта и его реализации на современных аппаратных платформах.

Алгоритмы приведения по простому модулю и их модификации

На начальном этапе развития вычислительной техники, был предложен классический алгоритм приведения по модулю, позаимствованный из школьной арифметики, который получил довольно широкое распространение.

Алгоритм 1. Классическое приведение целых чисел по простому модулю

Вход: целое $a = d \cdot e$, $d, e \in GF(p)$, $0 \le a < b^{n+m}$, $n = \lceil \log_b p \rceil$, $n \ge 2$, $m \ge 1$, $b = 2^w$, $w = 32$, $p_{n-1} \ge \frac{b}{2}$, p -простое.

Выход: $r = a \bmod p$

1. $r \leftarrow a$.

2. If $\left(r \ge p \cdot b^m\right)$

2.1. $r \leftarrow r - p \cdot b^m$.

3. For $i \leftarrow n + m + 1,\ i \ge n,\ i--$

3.1. If $\left(r_i = p_{n-1}\right)$

3.1.1. $q \leftarrow b - 1$.

3.1.2. Else // $r_i < p_{n-1}$

3.1.3. $q \leftarrow \left(r_i \cdot b + r_{i-1}\right) \operatorname{div} p_{n-1}$.

3.2. $y \leftarrow q \cdot \left(p_{n-1} \cdot b + p_{n-2}\right)$.

3.3. While $\left(y > r_i \cdot b^2 + r_{i-1} \cdot b + r_{i-2}\right)$ do

3.3.1. $q \leftarrow q - 1$.

3.3.2. $y \leftarrow y - (p_{n-1} \cdot b + p_{n-2})$.

3.4. $r \leftarrow r - q \cdot p \cdot b^{i-n}$.

3.5 If $(r < 0)$

3.5.1. $r \leftarrow r + p \cdot b^{i-n}$.

4. Return (r).

Вскоре было обнаружено, что программная реализация алгоритма «деления в столбик» является довольно медленной, что послужило толчком к поискам новых алгоритмов [1, 4]. Так появились алгоритмы Барретта [1] и Монтгомери [4].

В основу алгоритма приведения по модулю Монтгомери [4], положено деление на делитель специального вида, являющегося степенью b, вместо обычного деления на делитель общего вида. Пусть $R > p$ является целым, взаимно простым с модулем p, таким, что приведение по модулю R вычисляется легко. Классическим выбором является $R = b^k$. В этом методе, числа $u < p$, представлены p-вычетами соответствующими R, т.е. $uR \bmod p$. Такое представление целых чисел часто называют формой Монтгомери. Приведение Монтгомери числа u определяется как $uR^{-1} \bmod p$, где R^{-1} является мультипликативной инверсией R по модулю p. Это приведение предполагает, что целые числа представлены в форме Монтгомери. Пусть два целых числа $d < p$ и $e < p$ представлены в форме Монтгомери $dR \bmod p$ и $eR \bmod p$, соответственно, тогда их произведение имеет вид $deR^2 \bmod p$. Дальнейшее применение Алгоритма 2, позволит получить $r = deR \bmod p$, которое также находиться в форме Монтгомери. Так в алгоритме используется предвычисленная величина $\beta = -p^{-1} \bmod R$, которое после оптимизации [19] было сведено лишь к $\beta = -p_0^{-1} \bmod b$. В основу оптимизации положено наблюдение, что в основе алгоритма 2 лежит накопление сумы произведений p и r, до тех пор, пока результат станет кратным R. Такой эффект может быть достигнут посредством вычисления $r_i \beta \bmod b$, вместо $r_i p_0 \bmod R$, где $\beta = -p_0^{-1} \bmod b$.

Для проверки корректности алгоритма, заменим оригинальное значение r на $\bar{r}$. Так, $\beta \cdot p_0^{-1} = -1 \bmod b$, каждый раз в п.2.2 решается уравнение

$pt = pr_i\beta \equiv r_i \bmod b$. Заметим, что приведение требует не более одного финального вычитания p.

Алгоритм 2. Приведение Монтгомери целых чисел по простому модулю

Вход: целое $a = d \cdot e$, $d, e \in GF(p)$, $0 \le a < p \cdot b^n$, $n = \lceil \log_b p \rceil$, $n \ge 2$, $m \ge 1$, $b = 2^w$, $w = 32$, $\beta = -p_0^{-1} \bmod b$, p -простое.

Выход: $r = a(b^n)^{-1} \bmod p$.

1. $r \leftarrow a$.
2. For $i \leftarrow 0$, $i < n$, $i{+}{+}$.
2.1. $t \leftarrow (r_i \cdot \beta) \bmod b$.
2.2. $r \leftarrow r + (p \cdot t) \cdot b^i$.
3. $r \leftarrow r \operatorname{div} b^n$.
4. If $(r \ge p)$
4.1. $r \leftarrow r - p$.
5. Return (r).

Существует ряд модификаций алгоритма Монтгомери, рассмотрение которых выходит за рамки данной работы.

Описание алгоритма-прототипа Барретта приведения по простому модулю и его модификация

Алгоритм Барретта [1] основывается на идее положенной в основу арифметики с фиксированной запятой. Основной идеей является возможность оценить частное x/p , с помощью операций, которые могут быть предвычислены или менее затратные с вычислительной точки зрения, чем деление многократной точности. Остаток r от деления x/p может быть вычислен как $r = x - p\lfloor x/p \rfloor$. Учитывая этот факт, предположим, что деление на число b в некоторой степени:

$$r = x - p\left\lfloor \frac{\frac{x}{b^{n-1}} \frac{b^{2n}}{p}}{b^{n+1}} \right\rfloor = x - p\left\lfloor \frac{\frac{x}{b^{n-1}} \mu}{b^{n+1}} \right\rfloor$$, где $\mu = \left\lfloor \frac{b^{2n}}{p} \right\rfloor$ - заранее вычисленная константа.

Пусть $\hat{q}$ является оценкой частного x/p , в методе приведения, предложенном Барреттом, используется исключительно умножение многократной точности и сдвиги на размер машинного слова (обычное копирование). Тогда оценка остатку $\hat{r}$ от деления x/p можно записать:

$\hat{r} = \left(x \bmod b^{n+1} - \left(p\hat{q} \bmod b^{n+1}\right)\right) \bmod b^{n+1}$.

Оценка $\hat{r}$ подразумевает, что достаточно не более двух вычитаний модуля p для получения точного значения остатка r.

Алгоритм 3. Приведение Барретта целых чисел по простому модулю

Вход: целое $a = d \cdot e$, $d, e \in GF(p)$, $0 \le a < b^{2k}$, $k = \lfloor \log_b p \rfloor + 1$, $n = \left| \log_{2^w} p \right|$, $b = 2^w$, $w = 32$, p-простое, $\mu = \left\lfloor \frac{b^{2k}}{p} \right\rfloor$.

Выход: $r = a \bmod p$.

1. $\hat{q} \leftarrow \left\lfloor \left(\left(a \operatorname{div} b^{k-1} \right) \cdot \mu \right) \operatorname{div} b^{k+1} \right\rfloor$.

2. $r_1 \leftarrow \left(x \bmod b^{k+1} \right)$.

3. $r_2 \leftarrow \left(\hat{q} \cdot p \bmod b^{k+1} \right)$.

4. If $(r_1 > r_2)$

4.1. $r \leftarrow r_1 - r_2$.

4.3. Else

4.4. $r \leftarrow r_1 + b^{k+1} - r_2$.

5. While $(r \ge p)$

5.1. $r \leftarrow r - p$.

6. Return (r).

Для вычисления необходимо выполнить предварительный расчет константы $\mu = \left\lfloor \frac{b^{2k}}{p} \right\rfloor$, который производиться на этапе инициализации библиотеки.

Алгоритм 4. Расчет константы μ

Вход: $k = \lfloor \log_b p \rfloor + 1$, $n = \left| \log_{2^w} p \right|$, $b = 2^w$, $w = 32$, p-простое.

Выход: $\mu = \left\lfloor \frac{b^{2k}}{p} \right\rfloor$.

1. $\mu = \left\lfloor b^{2k} \operatorname{div} p \right\rfloor$.

2. Return (μ).

Известно, что в алгоритме Барретта в п. 1 и п. 3 выполняется два частичных (не полных) умножения больших целых чисел (Рис. 3).

Данная особенность позволяет воспользоваться ранее предложенными авторами модификациями алгоритма умножения ModComba [5]. Данные модификации являются урезанными версиями алгоритма умножения ModComba, т.к. в алгоритме Барретта нет необходимости в полном умножении,

т.к. используется либо старшая, либо младшая часть произведения. Применение урезанных версий алгоритма умножения ModComba, позволит повысить производительность приведения по модулю, а также откроет возможность дальнейшего повышения эффективности за счет применения методов распараллеливания предложенных авторами [5].

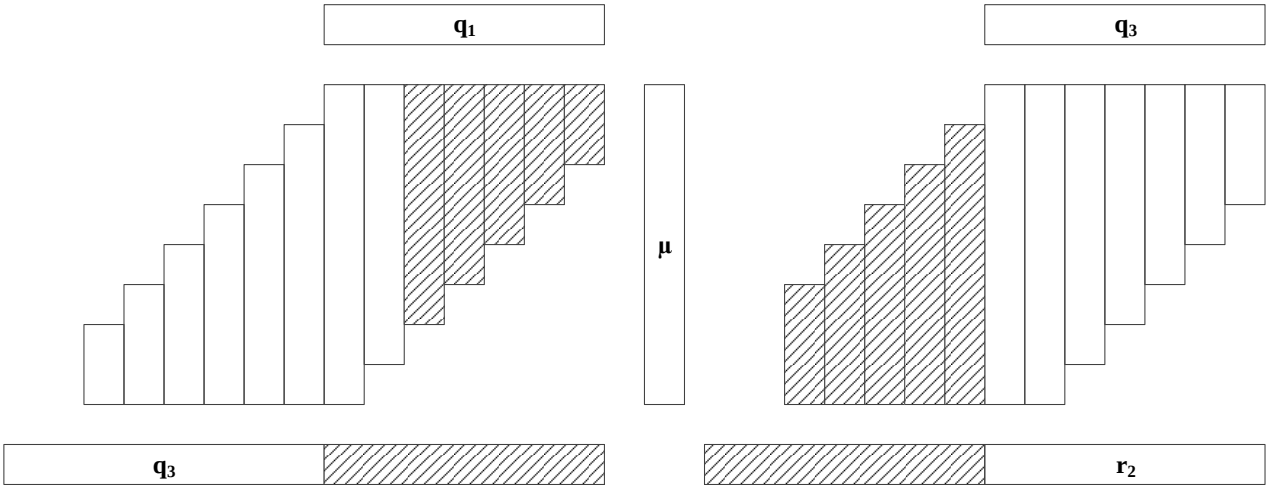

Рис. 3. Визуализация частичного умножения для алгоритма Барретта

Ниже, более подробно остановимся на проведенных модификация алгоритма Барретта. Как показано в алгоритме 5, для выполнения умножения п. 1 необходимо выполнить умножение двух больших чисел длиной n_2 и $\max(kb+k;n_2)+kb-1$ машинных слов, полученное произведение будет иметь длину $n_2+\max(kb+k;n_2)+kb-1$ машинных слов. Умножение больших чисел выполняется с помощью предложенного авторами [5] алгоритма ModComba, что позволит уменьшить объемы вычислений. В п.2 алгоритма 5, необходимо выполнить умножение двух больших чисел длинной n_2 и $n_3 \leftarrow n+n_2$.

Алгоритм 5. Приведение Барретта целых чисел по простому модулю с урезанным умножением ModComba

Вход: целое $a=d\cdot b$, $d,b\in GF(p)$, $w=32$, $n_2=\log_{2^w} a$, $n=\log_{2^w} d=\log_{2^w} b$, $n_3=n+n_2$.

Выход: $c=a \bmod p$, $c\in GF(p)$

1. $r_0^{(64)}\leftarrow 0$, $r_1^{(64)}\leftarrow 0$, $r_2^{(64)}\leftarrow 0$.

2. For $k\leftarrow 0$, $k<n_2$, $k++$ do

2.1. $kn_2\leftarrow \max(kb+k;n_2)$

2.2. For $i\leftarrow kb-1$, $j\leftarrow k$, $i<kn_2$, $i++$, $j--$ do

2.2.1. $(uv)^{(64)}\leftarrow a_i^{(32)}\cdot \mu_j^{32}$.

2.2.2. $r_0^{(64)} \leftarrow r_0^{(64)} + v^{(32)},\ r_1^{(64)} \leftarrow r_1^{(64)} + u^{(32)}.$

2.3. $r_1^{(64)} \leftarrow r_1^{(64)} + hi_{(32)}\left(r_0^{(64)}\right),\ r_2^{(64)} \leftarrow r_2^{(64)} + hi_{(32)}\left(r_1^{(64)}\right).$

2.4. $q_k^{(32)} \leftarrow low_{(32)}\left(r_0^{(64)}\right),\ r_0^{(64)} \leftarrow low_{(32)}\left(r_1^{(64)}\right),\ r_1^{(64)} \leftarrow low_{(32)}\left(r_2^{(64)}\right),\ r_2^{(64)} \leftarrow 0.$

3. For $k \leftarrow n_2,\ l \leftarrow kb, k < n_3,\ k++,\ l++$ do

3.1. For $i \leftarrow l,\ j \leftarrow n_2 - 1,\ i < n_2,\ i++,\ j--$ do

3.1.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot \mu_j^{32}.$

3.1.2. $r_0^{(64)} \leftarrow r_0^{(64)} + v^{(32)},\ r_1^{(64)} \leftarrow r_1^{(64)} + u^{(32)}.$

3.2. $r_1^{(64)} \leftarrow r_1^{(64)} + hi_{(32)}\left(r_0^{(64)}\right),\ r_2^{(64)} \leftarrow r_2^{(64)} + hi_{(32)}\left(r_1^{(64)}\right).$

3.3. $q_k^{(32)} \leftarrow low_{(32)}\left(r_0^{(64)}\right),\ r_0^{(64)} \leftarrow low_{(32)}\left(r_1^{(64)}\right),\ r_1^{(64)} \leftarrow low_{(32)}\left(r_2^{(64)}\right),\ r_2^{(64)} \leftarrow 0.$

4. $r_0^{(64)} \leftarrow 0,\ r_1^{(64)} \leftarrow 0,\ r_2^{(64)} \leftarrow 0.$

5. For $k \leftarrow 0,\ k < n,\ k++$ do

5.1. For $i \leftarrow 0,\ j \leftarrow k,\ i \le k,\ i++,\ j--$ do

5.2.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot \mu_j^{32}.$

5.2.2. $r_0^{(64)} \leftarrow r_0^{(64)} + v^{(32)},\ r_1^{(64)} \leftarrow r_1^{(64)} + u^{(32)}.$

5.3. $r_1^{(64)} \leftarrow r_1^{(64)} + hi_{(32)}\left(r_0^{(64)}\right),\ r_2^{(64)} \leftarrow r_2^{(64)} + hi_{(32)}\left(r_1^{(64)}\right).$

5.4. $q_k^{(32)} \leftarrow low_{(32)}\left(r_0^{(64)}\right),\ r_0^{(64)} \leftarrow low_{(32)}\left(r_1^{(64)}\right),\ r_1^{(64)} \leftarrow low_{(32)}\left(r_2^{(64)}\right),\ r_2^{(64)} \leftarrow 0.$

6. For $k \leftarrow n,\ l \leftarrow 1, k < kb + 1,\ k++,\ l++$ do

6.1. For $i \leftarrow l,\ j \leftarrow n - 1,\ i < n,\ i++,\ j--$ do

6.1.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot \mu_j^{32}.$

6.1.2. $r_0^{(64)} \leftarrow r_0^{(64)} + v^{(32)},\ r_1^{(64)} \leftarrow r_1^{(64)} + u^{(32)}.$

6.2. $r_1^{(64)} \leftarrow r_1^{(64)} + hi_{(32)}\left(r_0^{(64)}\right),\ r_2^{(64)} \leftarrow r_2^{(64)} + hi_{(32)}\left(r_1^{(64)}\right).$

6.3. $q_k^{(32)} \leftarrow low_{(32)}\left(r_0^{(64)}\right),\ r_0^{(64)} \leftarrow low_{(32)}\left(r_1^{(64)}\right),\ r_1^{(64)} \leftarrow low_{(32)}\left(r_2^{(64)}\right),\ r_2^{(64)} \leftarrow 0.$

7. $q_{kb+1}^{(32)} \leftarrow low_{(32)}\left(r_0^{(64)}\right).$

8. If $(q > a)$

8. 1. $c \leftarrow a + b^{kb+1} - q$

8. 2. Else

8. 3. $c \leftarrow a - q.$

9. While $(c \ge p)$ do

9.1. $c \leftarrow c - p$

10. Return (c).

В алгоритме 5, используются функции $\mathrm{hi}_{32}\left(r^{(64)}\right)$ и $\mathrm{low}_{32}\left(r^{(64)}\right)$ для выделения старшей и младшей 32-х разрядных частей 64-х разрядной переменной $r^{(64)}$, соответственно.

Сравнение с другими методами

Сравнения эффективности предложенных и известных алгоритмов оценивалось по среднему времени выполнения операции приведения по модулю, в разработанных тестовых приложениях. Усреднение времени проводилось для 1 млн. операций для простых чисел различной длины.

Предложенные алгоритмы были реализованы на языке высокого уровня C++ и скомпилированы с помощью Microsoft Intel C++ Compiler XE 2013 в Release x86 конфигурации с параметром /O3 с поддержкой SSE4.2. Для реализации параллельного выполнения, была использована библиотека OpenMP 2.0.

В качестве эталона, была выбрана библиотека GMP 5.1.3 *x86* с реализацией на C++ (для платформы generic no-asm), скомпилированная на основе исходных кодов с помощью msys динамическая библиотека libgmp.dll, которая затем была преобразована в статическую библиотеку с помощью утилиты lib из дистрибутива Visual Studio 2010. Тестовое приложение было построено с помощью Intel C++ Compiler XE 2013 в Release x86 конфигурации с параметром /O3 с поддержкой SSE4.2.

Также была собрана версия GMP 5.1.3 *x86* с реализацией на C++ (для платформы generic no-asm), скомпилированная на основе исходных кодов с помощью MinGW компилятора в Release x86 конфигурации. Известно, что в библиотеке GMP используется алгоритм приведения по модулю Монтгомери.

Эксперименты проводились на компьютерах с процессорами, приведенными в Таблице 1.

Результаты экспериментов – среднее время выполнения на настольной версии процессора Intel Core i5-3570 и на спаренной серверной версии процессора Intel Xeon E2640, приводиться в Таблице 3. Выделенные ячейки показывают случаи, в которых программные реализации предложенных алгоритмов показали себя хуже, чем в реализациях GMP.

Приведение по модулю большого простого числа, использовались простые числа, приведенные в Таблице 2, некоторые из них представлены в сокращенной форме (указаны начало и конец).

Таблица 1

Используемые в экспериментах процессоры

№	Обозначение процессора	Модель процессора	Число потоков
1	Core i5	Intel Core i5-3570	4
2	2 x Xeon	2 x Intel Xeon E5-2640	24

Таблица 2

Используемые простые числа

№	Обозначение	Простое число
1	p_{128}	340282366920938463463374607431768211507
2	p_{256}	115792089237316195423570985008687907……4007913129639747
3	p_{512}	134078079299425970995740249982058407……6433649006084171
4	p_{1024}	179769313486231590772930519078902407……6329624224137111
5	p_{2048}	323170060713110073007148766886699507……8536110595962316
6	p_{3072}	580960599536995806285950253330457407……4462567329693649
7	p_{4096}	104438888141315250669175271071662407……8340403154192097
8	p_{6144}	337515218214385611849111744886826407……4453197182135259
9	p_{8192}	109074813561941592946298424473378207……6654757154790457
10	p_{12288}	113916522526304337084593857931593207……3503514077625253
11	p_{16384}	118973149535723176508575932662800707……7290689699640530
12	p_{32768}	141546103104495478900155302774495107……1046343371235250

Из Таблицы 3 видно, что предложенный алгоритм приведения по модулю Барретта (MB – ModBarrett) с использованием алгоритма умножения целых чисел Комба с отложенным переносом, показал лучшие результаты на целых числах криптографической длины, до 8192 бит на Core i5 и до 4096 бит на Xeon E2640. Такое резкое отличие в производительности Core i5 по сравнению с Xeon E2640 связано с тем, что у Core i5 тактовая частота и производительность одного ядра выше.

Предложенный алгоритм приведения по модулю Барретта, с распараллеливанием в два потока (MB 2x – ModBarrett 2x) и использованием алгоритма умножения целых чисел ModComba с отложенным переносом и распараллеливанием в 2 потока, показал лучшие результаты на целых числах криптографической длины, с 1024 до 8192 бит на Core i5 и с 2048 до 8192 бит на Xeon E2640.

Таблица 3

Время выполнения операции приведения по модулю

Поле	Время, мкс									
	Core i5-3570					2 x Xeon E2640				
	MB	MB 2x	MB Mx	GMP static 5.1.3	GMP MinGW 5.1.3	MB	MB 2x	MB Mx	GMP static 5.1.3	GMP MinGW 5.1.3
p_{128}	0,00021	0,00134	0,00214	0,00024	0,00023	0,00031	0,00344	0,0128	0,00056	0,00033
p_{256}	0,00042	0,00120	0,00220	0,00044	0,00043	0,00094	0,00203	0,01248	0,00103	0,00094
p_{512}	0,00142	0,00206	0,00298	0,00163	0,00163	0,00187	0,00500	0,01575	0,00256	0,00226
p_{1024}	0,00424	0,00495	0,00464	0,00551	0,00539	0,00702	0,0109	0,01622	0,00968	0,0094
p_{2048}	0,0156	0,01725	0,0120	0,01983	0,01979	0,02683	0,02589	0,01966	0,02875	0,02816
p_{3072}	0,03443	0,03745	0,02925	0,04252	0,04211	0,05694	0,04836	0,02324	0,05936	0,06119
p_{4096}	0,05756	0,06231	0,02987	0,06517	0,06508	0,10983	0,09063	0,02902	0,09387	0,09429
p_{6144}	0,12583	0,13651	0,06768	0,14068	0,13990	0,21762	0,19203	0,04275	0,20149	0,20021
p_{8192}	0,2203	0,24022	0,11847	0,20983	0,20926	0,30779	0,33524	0,06879	0,29802	0,29929
p_{12288}	0,48869	0,53335	0,23616	0,42033	0,41916	0,71667	0,71495	0,11934	0,58853	0,58508
p_{16384}	0,86027	0,94044	0,43730	0,65414	0,65178	1,29110	1,27967	0,19157	0,92115	0,91522
p_{32768}	3,40335	3,7344	1,67451	1,80768	1,81818	4,96112	5,07157	0,69357	2,60224	2,59914

Такое резкое отличие в производительности Core i5 по сравнению с Xeon E2640 связано с тем, что у Core i5 тактовая частота и производительность одного ядра выше. С другой стороны, эффективность 2-х поточной версии начинает проявляться, с 1024 для Core i5 и с 2048 для Xeon E2640, соответственно, в связи с накладными расходами необходимыми для создания 2-х потоков и их синхронизации. При этом, на Xeon E2640 были задействованы лишь 2 ядра, а остальные 22 ядра – простаивали.

Предложенный алгоритм приведения по модулю Барретта с распараллеливанием на множестве потоков (MB Mx – ModBarrett Mx) использованием алгоритма умножения целых чисел ModComba с отложенным переносом и распараллеливание на множестве потоков, показал лучшие результаты на целых числах криптографической длины, с 1024 бит на Core i5 и с 2048 бит на Xeon E2640. Такое резкое отличие в производительности Core i5 по сравнению с Xeon E2640 связано с тем, что у Core i5 тактовая частота и производительность одного ядра выше. С другой стороны, эффективность 2-х поточной версии начинает проявляться, с 1024 для Core i5 и с 2048 для Xeon E2640, соответственно, в связи с накладными расходами необходимыми для создания 4-х и 24-х потоков и их синхронизации, соответственно.

Заключение

В работе произведен анализ известных методов представления целых чисел, а также известных методов приведения целых чисел по простому модулю, что позволило классифицировать их по различным характеристикам. Предложенная классификация позволила сформулировать основные направления дальнейшего совершенствования программной реализации алгоритмов приведения по модулю, а также предложить модифицированный алгоритм Барретта основанный на модифицированном алгоритме умножения ModComba с отложенным переносом, для эффективной программной реализации, оптимизированной для одно-, двух- и многоядерных процессоров.

Полученные алгоритмы и их программные реализации позволяют существенно повысить производительность криптографических преобразований с открытым ключом, основанных на операциях с целыми числами.

Литература

1. Barrett P. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor // Proceedings CRYPTO'86. – P. 311-323.

2. Hars Laszlo. Long Modular Multiplication for Cryptographic Applications // Cryptographic Hardware and Embedded Systems (CHES 2004). Lecture Notes in Computer Science Volume 3156. – 2004, P.45-61

3. Menezes Alfred, Handbook of Applied Cryptography. / Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone // CRC Press. – 2001. – 780 p.

4. Montgomery P. Modular multiplication without trial division. // Mathematics of Computation. – 44(170). – 1985. – P. 519–521.

5. Ковтун В.Ю., Охрименко А.А. Умножения целых чисел с использованием отложеного переноса для криптосистем с открытым ключом. – Информационные технологи и системы в управлении, образовании, науке: Монография / Под ред.. проф. В.С. Пономаренко. – Х.: Цифрова друкарня №1, 2013.– С. 69-82. – ISBN978-617-7017-37-9.