УДК 004.4 (043.2)

Самофалов К. Г., чл.-кор. НАН України,
Рамзи Анвар Салиба Сунна,
Романовский А. Е.

АЛГОРИТМ УСКОРЕННОГО МОДУЛЯРНОГО УМНОЖЕНИЯ ЧИСЕЛ БОЛЬШОЙ РАЗРЯДНОСТИ ПРИ ФИКСИРОВАННОМ МОДУЛЕ

Национальный технический университет Украины "КПИ"

Предложен новый алгоритм программной реализации модулярного умножения, использующий предвычисления при постоянном модуле. Разработанный алгоритм модулярного умножения обеспечивает большую производительность по сравнению с известными алгоритмами, ориентированными на переменное значение модуля, особенно при программной реализации на малоразрядных микроконтроллерах и смарт-картах. В этом случае предложенный алгоритм обеспечивает примерно вдвое большую производительность по сравнению с алгоритмом Монтгомери при относительно небольших затратах дополнительной памяти.

Введение

Операции модулярного умножения и модулярного возведения степень являются вычислительной основой важного класса современных алгоритмов защиты информации, базирующихся на теории чисел. В частности, вычислительная реализация таких важных для современных технологий обеспечения информационной безопасности механизмов, как: алгоритмы асимметричного шифрования (*RSA*, *ECC*), алгоритм обмена ключами *Diffie-Hellman*, алгоритм цифровой подписи *Digital Signsture Standard* основана на операциях модульного потенцирования и модульного умножения [2, 5].

Для обеспечения приемлемого уровня защищенности от вскрытий указанных выше алгоритмов, разрядность используемых в них чисел должна составлять тысячи бит. Так, для большинства применений, обеспечение информационной безопасности в современных условиях требует, чтобы разрядность чисел, используемых в алгоритмах на основе эллиптических кривых (*ECC*), составляла от 128 до 256, а алгоритмах, базирующихся на экспоненциальных преобразованиях – от 1024 до 2048 [5].

Программная реализация операции модулярного умножения для чисел столь большой разрядности на процессорах общего назначения и микроконтроллерах фиксированной разрядности (8, 16 и 32) требует значительных затрат времени.

Соответственно, программная реализация алгоритмов защиты информации с открытым ключом на универсальных процессорах выполняется на несколько порядков медленнее по сравнению с одноключевыми алгоритмами, такими как *DES* или *AES*, при условии обеспечения близкого уровня защищенности от вскрытий. Особенно остро проблема скорости выполнения операций модулярной арифметики больших чисел при реализации алгоритмов защиты информации стоит для малоразрядных встроенных микроконтроллеров [4]. Основной операцией модулярной арифметики, используемой в алгоритмах рассматриваемого класса, является модулярное потенцирование, то есть вычисление $A^K \ mod \ M$. При этом, в большинстве случаев, модулярное потенцирование выполняется методом "возведения в квадрат и умножения" [1], который использует число умножений близкое к теоретическому минимуму. Исходя из этого, основным резервом повышения производительности программной реализации рассматриваемых алгоритмов является уменьшения затрат времени на вы-

полнение операции модульного умножения.

Следовательно, важной и актуальной проблемой является поиск возможностей для повышения производительности программной реализации операций модульного умножения на процессорах общего назначения и микроконтроллерах.

Основные обозначения и модель оценки эффективности алгоритмов модулярного умножения

Базовой операцией модулярной арифметики, используемой в алгоритмах защиты информации является модулярное умножение, то есть вычисление $R=A\cdot B \bmod M$. Полагается, что результат R, множитель A, множимое B и модуль M представляют собой n-разрядные двоичные числа, причем, старший разряд модуля равен единице: $2^{n-1} \le M < 2^n$, а сомножители меньше модуля: $A<M$, $B<M$. Предполагается, что операция модулярного умножения выполняется на k-разрядном процессоре общего назначения (микропроцессоре или микроконтроллере). Соответственно, каждое из чисел, участвующих в операции модулярного умножения может быть представлено в виде $s=n/k$ –разрядных слов:

$$A=\sum_{j=0}^{s-1} a_j \cdot 2^{j\cdot k}, B=\sum_{j=0}^{s-1} b_j \cdot 2^{j\cdot k}, M=\sum_{j=0}^{s-1} m_j \cdot 2^{j\cdot k}$$

где a_j, b_j, m_j – k-разрядные слова, $j\in\{0,\ldots,s-1\}$.

В отличие от классического алгоритма [1] модулярного умножения, современные алгоритмы [3,6] не используют операции деления, которая неэффективно реализуется на процессорах общего назначения. Исходя из этого, в качестве критерия оценки производительности программной реализации алгоритмов модулярного умножения обычно рассматривается суммарное время выполнения операций умножения и сложения: основных операций современных алгоритмов модулярного умножения [5]. Полагается, что результатом команды умножения двух k-разрядных чисел является $2\cdot k$-разрядное произведение. Если обозначить через q_m – количество требуемых команд умножения (время выполнения каждой такой команды полагается равным t_m), а через q_a – количество команд сложения (при этом t_a – время выполнения каждой такой команды), то в качестве оценки времени выполнения модулярного умножения над n-разрядными числами можно с приемлемой для сравнительного анализа точностью считать $q_m \cdot t_m + q_a \cdot t_a$. Если полагать, что времена выполнения команд умножения и сложения на процессоре соотносятся как $w = t_{mul}/t_a$, то время выполнения модулярного умножения может быть представлено в следующем виде: $t_a \cdot (w \cdot q_m + q_a)$.

Краткий анализ современного состояния проблемы ускорения программной реализации модулярного умножения

Классический алгоритм программной реализации модулярного умножения [1] без детализации способа выполнения процедуры модулярной редукции *Reduce*(X) представлен в нотации языка *C++* на рис.1.

```
R=0;
for(i=0; i<s; i++)
{
  Y=0;
  for(j=0; j<s; j++)
  Y+= (a_i*b_j)<<(j*k);
  R += Reduce(Y);
  if (i<s-1)
  {
    B<<=k;
    Reduce(B);
  }
}
Reduce(R);
```

Рис. 1. Классическая схема пофрагментного модулярного умножения.

Операция умножения выполняется пофрагментно: каждое j-тое ($j=0,\ldots,s-1$) k–разрядное слово a_j множителя умножается со сдвигом на каждое из s слов множимого B, полученные $2\cdot k$-разрядные произведения слов суммируются, образуя $(n+k)$-разрядное частичное произведение $a_j \cdot B = \sum_{i=0}^{s-1} a_j \cdot b_i \cdot 2^{i \cdot k}$. Затем выполняется модулярная редукция полученного частичного произведения с получением j-того частичного остатка $R_j = a_j \cdot B \bmod M$. Результат модулярного умножения $R=A\cdot B \bmod M$ формируется в виде модулярной суммы остатков частичных произведений: $R=(R_0+R_1+\ldots+R_{s-1}) \bmod M$.

В классическом алгоритме модулярная редукция осуществляется с использованием операции целочисленного деления $2\cdot k$-разрядного делимого на k-разрядный делитель с получением частного и остатка. Поскольку деление n-разрядных чисел на k-разрядном процессоре ($n>>k$) выполняется весьма неэффективно, выполнение редукции в классическом алгоритме требует $s\cdot(s+2.5)$ операций умножения и s операций целочисленного деления [3]. К настоящему времени предложено ряд алгоритмов [2, 3, 5], повышающих производительность программной реализации операции модулярного умножения. Большая их часть реализует повышение производительности модулярного умножения за счет ускорения модулярной редукции путем исключения операции целочисленного деления, которое используется в классическом алгоритме [1]. К настоящему времени наиболее эффективным способом модулярного умножения является алгоритм Монтгомери [6], хорошо приспособленный к архитектуре универсального процессора. Алгоритм заменяет операцию деления на произвольный модуль M делениями на степень 2, которые эффективно реализуются сдвигами. Операция модулярной редукции в алгоритме Монтгомери требует $s\cdot(s+1)$ операций умножения.

Общая вычислительная сложность реализации алгоритма модулярного умножения Монтгомери на k-разрядном процессоре определяется $2\cdot s^2 + s$ операциями умножения и $4\cdot s^2+4\cdot s+2$ операциями сложения. Соответственно, время T_M выполнения алгоритма Монтгомери на k-разрядном процессоре приближенно может быть оценено следующим образом:

$$T_M = (2\cdot s^2+s)\cdot t_m + (4\cdot s^2+4\cdot s+2)\cdot t_a = \\ = t_a\cdot(s^2\cdot(2\cdot w+4)+s\cdot(w+4)+2). \quad (1)$$

Известные алгоритмы предполагают, что каждое вычисление модулярного умножения производится с новыми значениями сомножителей A,B и модуля M. Однако анализ практического применения алгоритмов защиты информации, использующих модулярное умножение показывает, что их ключи, а соответственно и модуль меняется относительно редко. Это открывает потенциальные возможности дальнейшего уменьшения вычислительной сложности модулярного умножения путем упрощения редукции. Практическая реализация таких возможностей требует специальных исследований и разработки на их основе новых алгоритмов модулярного умножения, которые бы учитывали постоянство модуля.

Целью настоящей работы является разработка эффективного алгоритма модулярного умножения больших чисел на процессорах общего назначения при постоянном модуле.

Анализ возможностей ускорения модулярного умножения в системах защиты информации

Алгоритмы защиты информации, в основе криптографических свойств которых лежат аналитически неразрешимые задачи теории чисел, требуют специальных сложных процедур генерации ключей. В частности, широко применяемый на практике алгоритм *RSA* [5] использует

сложную процедуру получения трех чисел d, e и M, разрядностью n от 1024 до 2048, которые удовлетворяют тождеству $A^{de} \equiv A \bmod M$. Процесс шифрования блока A сообщения состоит в вычислении $C=A^e \bmod M$, а дешифрования – в восстановлении исходного блока $A=C^d \bmod M$. Пара чисел $<d,M>$ составляет закрывающий ключ, а пара $<e,M>$ – открывающий. Один из упомянутых ключей в зависимости от протокола использования *RSA* является открытым, а второй держится в секрете. Анализ практики использования алгоритма *RSA* показывает, что ключи меняются относительно редко, так, что с использованием одного и того же ключа обрабатываются десятки тысяч информационных блоков. Это позволяет считать, что в процессе вычислительной реализации *RSA* ключ, а следовательно и модуль являются практически постоянными. Аналогичные рассуждения могут быть применены и к ряду других, стандартизированных и широко используемых на практике алгоритмов защиты информации, в частности к алгоритму цифровой подписи *Digital Signsture Standard* [5].

Постоянство модуля M позволяет упростить выполнение модулярной редукции в процессе умножения за счет использования результатов предвычислений. Такие предвычисления зависят только от значения модуля M, а поэтому выполняются однократно при изменении модуля. Результаты предвычислений сохраняются в табличной памяти и используются многократно при каждом выполнении модулярного умножения.

При реализации модулярного умножения, часть вычислительных ресурсов используется на выполнение собственно умножения, а другая часть – на нахождение остатка. В различных алгоритмах модулярного умножения удельный вес затрат на эти две процедуры различен. В таблице 1 приведено количество операции умножения и деления слов, используемое в наиболее известных алгоритмах модулярного умножения для вычисления произведения $A \cdot B$ и выполнения модулярной редукции [3].

Очевидно, что возможности уменьшения числа операций для вычисления произведения $A \cdot B$ за счет предвычислений при постоянном модуле весьма ограничены, поскольку сам модуль непосредственно не используется в таких вычислениях. Поэтому, основным резервом повышения скорости программной реализации модулярного умножения является использование предвычислений для снижения вычислительной сложности модулярной редукции.

Таблица 1. Количество операций умножения и деления над k-разрядными словами, используемых различными алгоритмами модулярного умножения для вычисления произведения $A \cdot B$ и остатка по модулю M

Алгоритм	К-во умножений k-разрядных слов для вычисления $A \cdot B$	К-во умножений k-разрядных слов для нахождения остатка	К-во делений слов для вычисления остатка
Классический	s^2	$s^2+2.5 \cdot s$	s
Баретта	s^2	$s^2+4 \cdot s$	0
Монтгомери	s^2	s^2+s	0

Анализ данных, приведенных в таблице 1 показывает, что наибольший эффект уменьшения времени реализации модулярного умножения за счет снижения затрат времени на вычисление остатка при использовании предвычислений достигается в рамках классического алгоритма.

Организация модулярного умножения на основе предвычислений при фиксированном модуле

В классическом алгоритме (рис.1) процедура модулярной редукции *Reduce*(X) выполняется над частичным произведением $a_j \cdot B$ и сдвинутым на k разрядов влево кодом множимого B. В обоих случаях, длина редуцируемого числа X не превышает $(s+1)$ k-разрядных слов или $(s+1)\cdot k=(n+k)$ двоичных разрядов x_0, x_1, $x_2,\ldots,x_{n+k-1}$:

$$X = \sum_{j=0}^{n+k-1} x_j \cdot 2^j, x_j \in \{0,1\}.$$

Число X может быть представлено в виде суммы двух компонент: $(n-1)$-разрядного числа X'', которое совпадает с $n-1$ младшими разрядами X и $(n+k)$-разрядного числа X', которое состоит из $(k+1)$ старших разрядов, совпадающих с одноименными разрядами X и $n-1$ младших разрядов, равных нулю:

$$X=\sum_{j=0}^{n+k-1} x_j \cdot 2^j = X'+X'', X'=\sum_{j=n-1}^{n+k-1} x_j \cdot 2^j, X''=\sum_{i=0}^{n-2} x_i \cdot 2^i.$$

В соответствии со свойством конгруэнтности для модулярной редукции, остаток $X \bmod M$ может быть представлен в виде модулярной редукции суммы остатков составляющих X компонент X' и X'':

$$X \bmod M = \left(\sum_{j=0}^{n+k-1} x_j \cdot 2^j\right) \bmod M =$$

$$= (X'+X'') \bmod M =$$

$$= (X' \bmod M + X'' \bmod M) \bmod M.$$

Поскольку старший, $(n-1)$-вый разряд модуля M равен единице, а X'' суть $(n-1)$-разрядное число, то $X''<M$ и, соответственно, $X'' \bmod M = X''$. Число X' содержит только $k+1$ значащих разрядов (остальные $n-1$ младших разрядов разны нулю). Следовательно, X' и соответственно $X' \bmod M$ принимает только 2^{k+1} различных значений. Все возможные n-разрядные значения $X' \bmod M$ для соответствующих X' могут быть заранее вычислены и сохранены в памяти в виде таблицы. Если обозначить через Z двоичный код, состоящий из $(k+1)$ старших значащих разрядов X': $Z = \sum_{j=n-1}^{n+k-1} x_j \cdot 2^{j-n+1}$ и через $T(Z)$ – n-разрядный код табличного значения $T(Z) = X' \bmod M$, то процедура модулярной редукции *Reduce*(X) реализуется в соответствии со следующим выражением:

$$\mathrm{Reduce}(X) = X \bmod M = (T(Z)+X'') \bmod M$$

Вычислительная сложность реализации модулярной редукции при этом определяется максимум двумя операциями типа сложения над $(n+k)$-разрядными числами: первого для вычисления $T(Z)+X''$ и второго для выполнения вычитания $(T(Z)+X'')-M$, если $T(Z)+X'' \geq M$. Поскольку $0 \leq T(Z)+X'' < 2 \cdot M$, для вычисления остатка $(T(Z)+X'') \bmod M$ потребуется не более одного вычитания n-разрядных чисел. Поэтому, время выполнения процедуры не превысит $2\cdot(s+1)\cdot t_a$, при среднем значении $1.5 \cdot s \cdot t_a$. Объем памяти, требующийся для хранения заранее вычисленных всех возможных значений $T(Z)$ составляет $2^{k+1} \cdot n$ бит или $2^{k+1} \cdot s$ k-разрядных слов.

Предлагаемый подход особенно эффективен при реализации модулярного умножения на малоразрядных микропроцессорах, микроконтроллерах и смарт-картах. В этом случае, объем памяти для хранения результатов предвычислений при постоянном модуле оказывается вполне приемлемым для большинства применений. Например, для реализации ускоренного умножения 1024-разрядных чисел на 8-разрядном микроконтроллере объем требуемой памяти составит всего $2^9 \cdot 128 = 2^{16}$ байт (64 Кбайт).

При реализации ускоренного модулярного умножения на процессорах разрядностью 16 и выше объем требуемой памяти существенно возрастает, что снижает эффективность применения предвычислений.

Для уменьшения объема памяти, требуемой для хранения результатов

предвычислений $T(Z)$ предлагается ее многосекционная организация.

Сущность предлагаемой организации таблиц предвычислений состоит в том, что значение X' разделяется на q составляющих: $X'=X_1'+X_2'+...+X_q'$ разрядностями $n+r_1$, $n+r_1+r_2$, ..., $n+r_1+...+r_q$, причем $r_1+r_2+...+r_q=k+1$. Каждая i–тая составляющая X'_i $(i=1,...,q)$ содержит r_i старших значащих разрядов, совпадающих с разрядами $x_{n+h}, x_{n+h+1},..., x_{n+h+r_i}$ числа X ($h=0$ для $i=1$ и $h=r_1+...+r_{i-1}$ для $i>1$), а остальные младшие разряды равны нулю:

$$X'_i=\sum_{h=g_i}^{g_i+r_i} x_{n+h}\cdot 2^{n+h}, g_i=\sum_{t=1}^{i-1} r_t, \forall i\in\{2,...,q\}, g_1=0.$$

Тогда, в соответствии со свойством конгруэнтности для модулярной редукции остаток $X \bmod M$ может быть представлен в виде:

$$X \bmod M=(X'_1 \bmod M+X'_2 \bmod M+...+$$
$$+X'_q \bmod M+X'') \bmod M.$$

Для определения каждого из значений $X'_i \bmod M$ предлагается использовать результаты предвычисления, которые осуществляются для всех возможных кодов X'_i. Поскольку количество значащих (ненулевых) разрядов в коде X'_i равно r_i, то число различных значений X'_i составит 2^{r_i}, а объем памяти для хранения всех возможных значений $X'_i \bmod M$ соответственно составит $2^{r_i}\cdot n$ бит. Если обозначить через Z_i двоичный r_i-разрядный код, содержащий только r_i старших значащих разрядов X'_i, то есть:

$$Z_i=\sum_{h=g_i}^{g_i+r_i} x_{n+h}\cdot 2^{h-g_i}, g_i=\sum_{t=1}^{i-1} r_t, \forall i\in\{2,...,q\}, g_1=0.$$

Если обозначить через $T_i(Z_i)$ – n-разрядный код табличного значения $T_i(Z_i) = X'_i \bmod M$, то процедура модулярной редукции реализуется в соответствии со следующим выражением:

$$X \bmod M=(\sum_{i=1}^{q} T_i(Z_i)+X'') \bmod M. \quad (2)$$

Общий объем табличной памяти для хранения $T_1(Z_1),T_2(Z_2),\ldots,T_q(Z_q)$ составляет $n\cdot\sum_{i=1}^{q} 2^{r_i}$ бит. Предложенный выше вариант хранения $T(Z)$ в одной таблице можно рассматривать, как частный случай секционированной организации таблиц результатов предвычислений при $q=1$.

Использование многосекционных таблиц позволяет существенно уменьшить объем памяти для их хранения. Например, в условиях приведенного выше примера реализации ускоренного умножения 1024-разрядных чисел на 8-разрядном микроконтроллере при двухсекционной памяти ($q=2$, $r_1=5$, $r_2=4$), объем требуемой памяти составит всего $1024\cdot(2^5+2^4)= 10^{10}\cdot 48$ бит или 6144 байт или в 10.67 раз меньше, чем при односекционной организации табличной памяти.

С другой стороны, использование многосекционной организации табличной памяти сопряжено с увеличением времени выполнения модулярной редукции. Вычисление суммы в выражении (2) требует $q\cdot(s+1)$ операций суммирования k-разрядных слов. Количество значащих разрядов кода суммы не превысит при этом $n+q$, так, что, если $r_1\geq q+1$, то для выполнения модулярной редукции суммы с использованием первой таблицы $T_1(Z_1)$ потребуется, в среднем, $1.5\cdot(s+1)$ операций типа сложения. Общее же количество операций типа сложения составит $(q+1.5)\cdot(s+1)$.

Оценка и сравнительный анализ производительности

Важным аспектом анализа эффективности предложенного алгоритма модулярного умножения является оценка времени программной реализации на k-разрядном процессоре и сравнение ее с наиболее эффективным на сегодняшний день алгоритмом Монтгомери.

Предлагаемый алгоритм ускоренного модулярного умножения с использованием предвычислений включает в себя s циклов. Каждый j-тый ($j=0,...,s-1$) из этих

циклов реализует вычисление $a_j \cdot B$, нахождение остатка $R = a_j \cdot B \bmod M$, сдвиг B на k разрядов с последующей модулярной редукцией для нахождением нового значения B: $B=B \cdot 2^k \bmod M$. Таким образом, процедура модулярной редукции на каждом цикле осуществляется дважды, что требует в сумме $3 \cdot (s+1)$ операций сложения слов. В свою очередь, вычисление частичного произведения $a_j \cdot B$ состоит из s умножений $a_j \cdot b_e$, где индекс e последовательно принимает значения от 0 до $s-1$, и, в среднем, 3-х операций сложения слов для прибавления $2 \cdot k$-разрядного результата умножения $a_j \cdot b_e$ к частичной сумме. Таким образом, программная реализация модулярного умножения в соответствии с предложенным алгоритмом на k-разрядном процессоре требует s^2 операций умножения слов и $3 \cdot s^2 + 7.5 \cdot s + 1.5$ операций сложения слов. При этом, реализация собственно умножения требует s^2 операций умножения слов и $3 \cdot s$ операций сложения, а модульная редукция - $3 \cdot s \cdot (s+1) + 1.5 \cdot (s+1)$ операций типа сложения. Соответственно, среднее время T_1 программной реализации модулярного умножения при использовании одной таблицы предвычислений определяется следующим выражением:

$$T_1 = s^2 \cdot t_m + (3 \cdot s^2 + 7.5 \cdot s + 1.5) \cdot t_a \approx \approx t_a \cdot (s^2 \cdot (3 + w) + 7.5 \cdot s). \quad (3)$$

По сравнению с алгоритмом Монтгомери (1), предлагаемый алгоритм с предвычислениями позволяет повысить скорость программной реализации модулярного умножения в d раз:

$$d = \frac{T_M}{T_1} = \frac{s^2 \cdot (2 \cdot w + 4) + s \cdot (w + 4) + 2}{s^2 \cdot (w + 3) + 7.5 \cdot s}.$$

Для наиболее распространенных на практике длин (от 1024 до 2048 бит) чисел, участвующих в модулярном умножении и малой разрядности процессора (k=8 бит), на котором оно реализуется, величина s достаточно велика, так, что $s^2 >> s$, и коэффициент d ускорения может быть, с достаточной для сравнительной оценки точностью, представлен в виде:

$$d \approx \frac{2 \cdot w + 4}{w + 3} \approx 2.$$

Результаты экспериментальных исследований показали, что использование предложенного алгоритма модулярного умножения с предвычислениями позволяет повысить производительность программной реализации на 8-разрядном контроллере по сравнению с алгоритмом Монтгомери примерно в 1.8 раз. При этом сложность программы реализации предложенного алгоритма примерно вдвое меньше по сравнению с программой, реализующей алгоритм Монтгомери.

При использовании q-секционной организации таблиц предвычислений время T_q модулярного умножения с использованием результатов предвычислений увеличивается по сравнению с T_1 и определяется следующим выражением:

$$T_q = s^2 \cdot t_m + (s^2 \cdot (2 \cdot q + 3) + s \cdot (q + 7.5) + q + 1.5) \cdot t_a \approx \approx t_a \cdot (s^2 \cdot (w + 2 \cdot q + 3) + s \cdot (q + 7.5)). \quad (4)$$

Очевидно, что выражение (3) для оценки времени T_1 модулярного умножения при использовании единой таблицы предвычислений является частным случаем выражения (4) при q=1.

Коэффициент d_q ускорения операции модулярного умножения по сравнении с алгоритмом Монтгомери для случая, если таблицы предвычислений организованы в виде q секций определяется в виде:

$$d_q = \frac{s^2 \cdot (2 \cdot w + 4) + s \cdot (w + 4)}{s^2 \cdot (w + 2 \cdot q + 3) + s \cdot (q + 7.5)}.$$

В качестве грубой оценки численного значения коэффициента d_q при больших значениях s, при этом $s^2 >> s$, можно учитывать только отношение компонент, входящих в выражения для T_M и T_q с множителем s^2:

$$d_q \approx \frac{2 \cdot w + 4}{w + 2 \cdot q + 3}. \quad (5)$$

Анализ выражения (5) показывает, что выигрыш в скорости реализации модулярного умножения по сравнению с алгоритмом Монтгомери при многосекци-

онной организации таблиц в определяющей степени зависит от значения w – отношения времени выполнения команд умножения и сложения. Выигрыш в скорости тем существеннее, чем больше значение w. В свою очередь, значение w зависит от разрядности k процессора и имеющихся в его составе аппаратных средств ускорения умножения.

Выводы

Исследована проблема повышения производительности программной реализации базовой вычислительной операции широкого круга алгоритмов защиты информации – модулярного умножения. Показано, что при практическом применении алгоритмов защиты информации, основанных на аналитически неразрешимых задачах теории чисел, ключи, а следовательно и модуль, меняются относительно редко.

На основе проведенных исследований предложен новый алгоритм модулярного умножения, отличающийся от классического организацией выполнения модулярной редукции. Снижение вычислительной сложности программной реализации достигается за счет использования результатов предвычислений, зависящих только от модуля и хранящихся в табличной памяти. Теоретически обоснованы оценки производительности предложенного алгоритма и затрат памяти для хранения таблиц предвычислений. Полученные оценки, в целом, подтверждаются результатами экспериментальных исследований. Определено, что наибольшая эффективность применения алгоритма достигается при реализации модулярного умножения на малоразрядных микроконтроллерах и смарт-картах.

Выполненный анализ показал, что скорость программной реализации модулярного умножения на микроконтроллерах при использовании предложенного алгоритма возрастает в 1.5-2 раза в сравнении с наиболее производительным на сегодняшний день алгоритмом Монтгомери.

Список литературы

1. *Кнут Д.* Искусство программирования для ЭВМ. Т.2. Получисленные алгоритмы. – М.: Мир, 1977. – 843 с.

2. *Харин Ю. С., Берник В. И., Матвеев Г. В., Агиевич С. В.* Математические и компьютерные основы криптологии / – М.: Новое знание, 2003. – 382 с.

3. *Bosselaers A., Govaerts R., Vandewalle J.* Comparison of three modular reduction functions // Proceeding of Advances in Cryptology CRYPTO'93, LNCS-773, Springer-Verlag, 1993. – P. 175-186.

4. *Dhem J.-F., Quisquater J.-J.* Resent results on modular multiplications for smart cards // Proceeding of GARDIS 1998. LNCS-1820, Springer-Verlag, 2000. – P. 350-366.

5. *Menezer A. J., Van Oorschot P. C., Vanstone S. A.* Handbook of Applied Cryptography. CRC-Press, 1997. – 780 с.

6. *Montgommery P. L.* Modular multiplication without trial division // Mathematics of Computation, 1985. – Vol. 44. – P. 519-521.