УДК 004.4 (043.2)

Самофалов К. Г., член-кор. НАН України,
Рамзи Анвар Салиба Сунна,
Левчун Д. Ю.

УСКОРЕННАЯ РЕАЛИЗАЦИЯ МОДУЛЯРНОГО ЭКСПОНЕНЦИРОВАНИЯ НА МАЛОРАЗРЯДНЫХ МИКРОПРОЦЕССОРАХ И ВСТРОЕННЫХ МИКРОКОНТРОЛЛЕРАХ

Национальный технический университет Украины "КПИ"

В статье предложены новые алгоритмы для модулярного возведения в квадрат и умножения на фиксированное число, основанные на рекурсии Монтгомери. Благодаря исключению избыточных операций и использованию предвычислений, вычислительная сложность предложенных алгоритмов существенно ниже по сравнению с алгоритмом умножения Монтгомери. Показано, что при реализации предложенных алгоритмов на малоразрядных микропроцессорах, встроенных микроконтроллерах или смарт-картах, время вычисления модулярного экспоненцирования в шесть раз меньше по сравнению с алгоритмом модулярного экспоненцирования Монтгомери.

Введение

Одной из доминирующих тенденций современного этапа развития информационных технологий является расширяющаяся информационная интеграция на основе компьютерных сетей. Обеспечивая качественно более высокий уровень решения прикладных задач, информационная интеграция, охватывает все уровни компьютерной обработки информации, включая встроенные микропроцессорные средства, которые уже в настоящее время составляют более половины терминальных устройств компьютерных сетей [1].

Важным условием эффективного взаимодействия вычислительных терминальных устройств в компьютерных сетях является достижение высокой производительности при реализации существующих сетевых протоколов. Для большей части микропроцессорных средств и встроенных микроконтроллеров, наиболее критичным, с точки зрения вычислительной реализации протоколов сетевого обмена, являются операции модулярной арифметики, выполняемые над целыми числами большой разрядности, во много раз превышающими разрядность процессора. Подобные вычисления предусмотрены, в частности, существующими протоколами аутентификации абонентов сети и обеспечения подлинности передаваемых по сети информационных сообщений.

Вычислительной основой существующих механизмов аутентификации удаленных абонентов компьютерных сетей, подтверждения подлинности и аутентичности сообщений является модулярное экспоненцирование, то есть вычисление $X^E \bmod M$. Разрядности чисел X, M и E в настоящее время составляют 2048 бит с перспективой увеличения в ближайшие годы до 4096 [2]. Вполне очевидно, что реализация столь громоздких вычислений на процессорах общего назначения, малоразрядных микропроцессорах, встроенных микроконтроллерах и смарт-картах требует существенных затрат времени, что снижает эффективность их работы в сети.

Таким образом, проблема повышения производительности программной реализации операций модулярной арифметики на процессорах общего назначения и встроенных микроконтроллерах является одной из наиболее актуальных и важных для современного этапа развития сетевых технологий.

Анализ современного состояния проблемы быстрой программной реализации модулярного экспоненцирования

Базовой вычислительной операцией при программной реализации большей части алгоритмов защиты информации с открытым ключом является модулярное экспоненцирование $X^E \bmod M$, причем $X, E < M$. Количество двоичных разрядов n чисел X, E и M существенно превышает разрядность k процессора. Соответственно, каждое из чисел X, E и M, может быть представлено в виде совокупности s k-разрядных двоичных чисел ($s=n/k$), каждое из которых лежит в интервале от 0 до 2^k-1: $X=\{x_{s-1},\ldots,x_1,x_0\}$, $E=\{e_{s-1},\ldots,e_1,e_0\}$ и $M=\{m_{s-1},\ldots,m_1,m_0\}$, $\forall j \in \{0,\ldots,s-1\}$: $0 \le x_j,e_j,m_j \le 2^k-1$:

$$X = \sum_{j=0}^{s-1} x_j \cdot 2^{j \cdot k}, E = \sum_{j=0}^{s-1} e_j \cdot 2^{j \cdot k},$$

$$M = \sum_{j=0}^{s-1} m_j \cdot 2^{j \cdot k}.$$

Вычислительную сложность алгоритмов модулярного экспоненцирования при их реализации на различных вычислительных платформах обычно оценивают количеством используемых в них операций процессорного умножения [4].

Процессорными будем называть целочисленные операции, выполняемые над k-разрядными числами, длина которых соответствует разрядности процессора. Процесс модулярного экспоненцирования сводится к последовательному выполнению $\log_2 E=n$ циклов, в каждом из которых осуществляется операция возведения в квадрат полученного на предшествующем цикле результата и, дополнительно, в зависимости от текущего бита степени E, выполняется операция умножения. Исходя из порядка, в котором анализируются разряды степени E можно рассматривать два типа алгоритмов экспоненцирования. На практике, большее распространение получили алгоритмы, которые предполагают анализ разрядов степени E, начиная со старших разрядов (слева направо): в этом случае множитель при выполнении операции умножения представляет собой постоянной число, равное X, что создает потенциальные предпосылки для повышения скорости умножения.

С использованием нотаций языка C++, алгоритм модулярного экспоненцирования этого типа может быть представлен в виде:

1. $A = 1$.
2. *for* ($j=n; j>=0; j$ --)
{
   2.2. $A = A \cdot A \bmod M$
   2.1. *if* ($e_j == 1$) $A = A \cdot X \bmod M$
}
3. Результат: $X^E \bmod M$.

Анализ приведенного алгоритма показывает, что базовыми операциями при выполнении модулярного экспоненцирования являются модулярное возведение в квадрат и модулярное умножение на фиксированное число, время выполнения которых, фактически определяет производительность вычисления $X^E \bmod M$.

В большинстве алгоритмов модулярного экспоненцирования для реализации упомянутых двух операций используется единая операция модулярного умножения [5]. В свою очередь, время выполнения модулярного умножения определяется двумя составляющими: временем, требующимся на реализацию собственно умножения и времени, затрачиваемого на модулярную редукцию. При классическом умножении модулярная редукция реализуется с использованием операции деления [2] и, соответственно, вторая составляющая играет доминирующую роль. Существенно большая эффективность вычислительной реализации модулярного умножения достигается при использовании алгоритма Монтгомери [6], в котором модулярная редукция сводится к сдвигу на k разрядов. Поэтому, в большинстве практических применений при выполнении модулярного экспоненцирования используется алгоритм Монтгомери [2]. Обозначим как $Mont(A,B)$ – умножение Монтгомери, которое формирует результат $R=A \cdot B \cdot U \bmod M$, где U – модулярная инверсия числа 2^n по модулю

M, то есть $U = (2^n)^{-1} \bmod M$. Алгоритм умножения Монтгомери – $Mont(A,B)$, может быть представлен с использованием нотаций C++ в следующем виде:

1. $R = 0$.
2. *for* (j=0; j <= s-1; j++)
   {
   2.1. $p_j = (r_0 + a_j \cdot b_0) \cdot M' \bmod 2^k$;
   2.2. $R = (R + a_j \cdot B + p_j \cdot M) >> k$;
   }
3. *if* ($R \geq M$) $R = R - M$,

где $M' = -M^{-1} \bmod 2^k$, причем значение M' определяется из условия $(M \cdot M^{-1}) \bmod U = 1$; $U = 2^n$. Результатом работа алгоритма является $Mont(X,Y) = A = X \cdot Y \cdot U^{-1} \bmod M$. Вычислительная сложность реализации алгоритма на k-разрядном процессоре определяется $2 \cdot s^2 + s$ процессорными операциями умножения и $4 \cdot s^2 + 4 \cdot s$ операциями сложения.

Алгоритм модулярного экспоненцирования Монтгомери можно представить с использований введенных нотаций следующим образом:

1. $\tilde{x} = Mont(X, U^2 \bmod M) = X \cdot U \bmod M, A = U \bmod M$
2. *for* ($j = n; j >= 0; j$ --)
   {
   2.1. $A = Mont(A,A)$;
   2.2. *if* ($e_j = 1$) $A \leftarrow Mont(A, \tilde{x})$
   }
3. $A \leftarrow Mont(A,1)$.

Для реализации первого шага этого алгоритма требуется одно умножение Монтгомери или $2 \cdot s \cdot (s+1)$ операций процессорного умножения.

Среднее число N_B операций процессорного умножения, используемых в представленном базовом алгоритме модулярного экспоненцирования Монтгомери определяется следующей формулой:

$$N_B = 3 \cdot s \cdot (s+1) \cdot (n+1). \qquad (1)$$

Для реализации каждого шага 2 алгоритма экспоненцирования Монтгомери требуется $3 \cdot s \cdot (s+1)$ процессорных умножений; при использовании классической схемы умножения на каждом шаге требуется такое же количество процессорных умножений и дополнительно s операций процессорного деления, время реализации которого для большинства вычислительных платформ существенно превышает время процессорного умножения.

К настоящему времени разработано ряд методов ускоренной реализации модулярного экспоненцирования [5], реализующие следующие возможности ускорения вычислений:

- уменьшение числа циклов при выполнении модулярного экспоненцирования за счет рациональной организации вычислений с использованием аддитивных цепочек [4];
- снижение времени выполнения операций модулярного умножения за счет обработки нескольких смежных разрядов множителя [3];
- ускорение возведения в квадрат за счет исключения избыточных операций процессорного умножения [3];
- уменьшение вычислительной сложности модулярного экспоненцирования за счет упрощений, имеющих место при учете специфических особенностей практического использования этой операции в алгоритмах защиты информации, применяемых в современных сетевых технологиях [1].

Последняя из приведенных потенциальных возможностей ускорения программной реализации модулярного экспоненцирования представляется наиболее перспективной.

Анализ практического использования алгоритмов защиты информации типа *RSA*, *El-Gamal*, алгоритма подтверждения подлинности и аутентичности – *DSA* [1], основанных на операции модулярного экспоненцирования свидетельствует о том, что их ключи меняются редко, поскольку их открытая часть идентифицирует абонентов сети. Так как модуль M и степень X однозначно определяются кодом ключа, то и они меняются относительно редко, так, что при организации вычислений можно их считать их значения постоянными [1]. С учетом этого,

время выполнения модулярного экспоненцирования может быть существенно уменьшено за счет широкого использования предвычислений, зависящих от модуля и степени.

Целью работы является исследование возможностей уменьшения времени выполнения операции модулярного экспоненцирования за счет учета особенностей ее практического использования в составе алгоритмов защиты информации и разработка на этой основе алгоритмов ускоренной программной реализации этой операции.

Алгоритм ускоренного возведения в квадрат на основе рекурсии Монтгомери

Проведенный анализ показал, что производительность программной реализации модулярного экспоненцирования может быть повышена путем разработки новых алгоритмов выполнения модулярного возведения в квадрат и модулярного умножения на постоянный множитель – базовых операций модулярного умножения. Эти алгоритмы должны:

- реализовать редукцию промежуточных результатов по способу Монтгомери;
- исключать дублирование операций, имеющее место при возведении в квадрат и умножении на постоянный множитель;
- использовать специфические особенности реализации модулярного экспоненцирования в рамках стандартизированных алгоритмов защиты информации в сетях. Эти особенности состоят в постоянстве модуля и степени, что позволяет широко использовать результаты предвычислений.

Анализ базового алгоритма Монтгомери показывает, что при вычислении $X^2 \bmod M$ имеет место избыточность, связанная с тем, что процессорные операции умножения $x_i \cdot x_j$ и $x_j \cdot x_i$, $i,j \in \{0,\ldots,s\text{-}1\}$, $i \neq j$, результаты которых равны, выполняются дважды. Эта избыточность исключается при вычислении $X^2 \bmod M$ в виде:

$$X^2 \bmod M = \sum_{i=0}^{s-2} \sum_{j=i+1}^{s-1} 2 \cdot x_i \cdot x_j \cdot 2^{i+j} + \\ + \sum_{j=0}^{s-1} x_j \cdot x_j \cdot 2^{2 \cdot j}. \qquad (2)$$

Подобный подход к повышению производительности операции модулярного возведения в квадрат предложен в работе [3], в которой разработаны алгоритмы модулярного возведения в квадрат, основанные на исключении избыточных операций процессорного умножения. В основе этих алгоритмов лежит рекурсивное вычисление модулярного возведения в квадрат в виде:

$$(X^{(j-1)})^2 \bmod M = (X^{(j)})^2 \cdot 2^{2 \cdot k} + \\ + 2 \cdot x_j \cdot X^{(j)} \cdot 2^k + x_j,$$

где

$$X^{(j)} = (\ldots((x_{s-1} \cdot 2^k + x_{s-2}) \cdot 2^k + \\ + x_{s-3}) \cdot 2^k + \ldots) \cdot 2^k + x_j.$$

Показано, что при таком порядке возведения в квадрат требуется меньшее число операций процессорного умножения, чем в алгоритме умножения Монтгомери. Однако, анализ показывает, что такой эффект достигается за счет дополнительного введения операций вычитаний, которые реализуют модулярную редукцию, причем их вычислительная сложность $O(2 \cdot s^3)$ достаточно велика. Кроме того, следует учесть, что кроме операции возведения в квадрат процедура модулярного экспоненцирования включает операцию обычного умножения, которую выгоднее выполнять по методу Монтгомери. Поэтому важным является разработка алгоритма возведения в квадрат без использования избыточных операций с использованием рекурсии Монтгомери.

С учетом выражения (2), вычисление $X^2 \bmod M$ на основе рекурсии Монтгомери может быть организовано в виде s-1 циклов. В каждом из циклов обрабатывается j-тый k-разрядный фрагмент x_j числа X: вычисляется поправка p Монтгомери, квадрат фрагмента x_j^2, произведение

фрагмента x_j на все фрагменты X со старшим весом $x_{j+1},\ldots,x_{s-1}$, которые прибавляются к значению текущей суммы частичных произведений R.

С использованием нотаций *C++*, предлагаемый алгоритм модулярного возведения в квадрат может быть представлен в следующем виде:

1. $R = x_0 \cdot x_0$; $d = x_0$;
2. *for* (j=1; j<=s-1; j++)
{
   2.1. $p = r_0 \cdot M' \bmod 2^k$
   2.2. $R=(R+ p \cdot M) >> k$;
   2.3. $X = X>>k$;
   2.4. $t = (2 \cdot d \cdot X)<<(k \cdot (j-1))$
   2.5. $z = (x_0 \cdot x_0)<<(k \cdot j)$
   2.6. $R = R + t + z$; $d = x_0$
}
3. $p = r_0 \cdot M' \bmod 2^k$
4. $R=(R + p \cdot M) >> k$;
5. *if* ($R >= M$) $R = R - M$

Результатом работы алгоритма является $R = X^2 \cdot U^{-1} \bmod M$.

Предложенный модифицированный алгоритм модулярного возведения в квадрат иллюстрируется следующим примером. Пусть выполняется модулярное возведение в квадрат $X^2 \bmod M$, на 4-разрядном процессоре (k=4), разрядность операндов равна 12 (n=12), то есть каждый из них состоит из 3-х процессорных слов (s=3), каждое из которых соответствует символу шестнадцатеричного их представления, причем X= 0x8*FD*, M=0x9*A*3. Соответственно, U=0x1000, U^{-1} = 0x868, M'=-M^{-1} mod 2^k =5. Результатом операции модулярного возведения в квадрат является код $X^2 \cdot U^{-1}$ mod M = 0x4*D*1.

Ход вычислений, значения переменных по циклам выполнения предложенного алгоритма модулярного возведения в квадрат для данного примера приведены в таблице 1.

Таблица 1

Цифровая диаграмма модулярного возведения в квадрат с использованием редукции Монтгомери

До цикла: x_0 =0x*D*, $A=x_0 \cdot x_0$ =0x*A*9								
j	a_0	u	$u \cdot M$	$A=(A+u \cdot M)>>k$	$X>>=k$	t	p	$A+=t+p$
1	0x9	0x*D*	0x7*D*47	0x7*DF*	0x8*F*	0x*E*86	0x*E*10	0x2475
2	0x5	0x9	0x56*BB*	0x7*B*3	0x8	0x*F*00	0x4000	0x56*B*3
После цикла: a_0=3, u=0x*F*, $u \cdot M$=0x908*D*, $A=(A+u \cdot M)>>k$=0x*E*74 Так как $A>M$, выполняется коррекция $A=A-M$=0x4*D*1								

Цикл выполняется s-1 раз. В каждом j-том цикле выполняется : одно умножение для вычисления $r_0 \cdot M'$, s операций процессорного умножения для вычисления $p \cdot M$, s-j операций умножения для вычисления $d \cdot X$, одно процессорное умножение для вычисления $x_j \cdot x_j$. Суммарное количество операций процессорного умножения, выполняемых в s-1циклах алгоритма, вычисляется как сумма арифметической прогрессии в виде (s-1)·(3·s+4)/2. До и после цикла выполняется еще 2+s умножений. Таким образом, общее число N_M операций процессорного умножения, используемых при реализации предложенного алгоритма составляет:

$$N_M = \frac{(s-1)\cdot(3\cdot s+4)}{2} + s + 2 = \quad (3)$$
$$= 1.5 \cdot s^2 + 1.5 \cdot s.$$

По сравнению с базовым модулярным умножением Монтгомери, предложенный алгоритм позволяет, за счет исключения избыточных процессорных операций, ускорить выполнение модулярного возведения в квадрат примерно в 1,3 раза.

На практике, модулярное экспоненцирование, чаще всего, используется для реализации алгоритмов защиты информации с открытым ключом, типа *RSA*, *El-Gamal*, алгоритмов цифровой подписи типа *DSA*. Ключ в таких алгоритмах, открытая часть которого распространяется

среди абонентов сети, меняется относительно редко [5]. Следовательно и модуль, определяемый кодом ключа, меняется достаточно редко. Для случая постоянного модуля, приведенный алгоритм может быть существенно упрощен за счет использования предвычислений, зависящих только от модуля. В частности, значение поправки p, вычисляемое в п.2.1 и произведение $p \cdot M$ (п.2.2.) зависят только от значения r_0 – младших k-разрядов суммы частичных произведений и при малых значениях разрядности процессора k (8,16) могут быть заранее вычислены для всех 2^k возможных значений r_0 в виде $T_1(r_0)$ с сохранением результатов в табличной памяти. При этом значение $T_1(r_0)$ вычисляется в соответствии с выражением:

$$T_1(r_0) = r_0 + M \cdot (r_0 \cdot M' \bmod 2^k).$$

Использование предвычислений позволит исключить s+1 операций процессорного умножения на каждом цикле, используемых для выполнения модулярной редукции. Соответственно, общее количество N_M' операций процессорного умножения в случае использования таблиц $T_1(r_0)$ определяется числом таких операций, необходимых только для возведения в квадрат:

$$N_M' = \frac{(s-1)\cdot(s+1)}{2} + 1. \qquad (4)$$

Анализ (4) показывает, что по сравнению с использованием модулярного умножения Монтгомери, число требуемых операций процессорного умножения при использовании предвычислений в рамках предложенного алгоритма модулярного возведения в квадрат уменьшается примерно в 4 раза.

Ускоренная реализация модулярного умножения на постоянный множитель

Еще одной, потенциальной возможностью ускорения программной реализации модулярного экспоненцирования является использование предвычислений при умножении на постоянный множитель.

Для оценки эффективности такого подхода исследуем возможности ускорения умножения по Монтгомери при условии постоянства одного из сомножителей. Анализ описанной выше базовой процедуры $Mont(A,B)$ показывает, при условии постоянства значения множимого – B, представляется возможным заранее (то есть перед каждым выполнением модулярного экспоненцирования), вычислить значения: $a_j \cdot b_0$ и $a_j \cdot B$ для всех 2^k возможных значений k-разрядного фрагмента множителя a_j. Это позволит уменьшить число операций процессорного умножения до величины $s^2 + s$ против $2 \cdot s^2 + 2 \cdot s$ в базовом алгоритме умножения Монтгомери.

С другой стороны, для выполнения таких предвычислений, необходимо выполнить $2^k \cdot s$ операций процессорного умножения. При выполнения модулярного экспоненцирования $X^E \bmod M$ операция модулярного умножения на постоянное число выполняется в среднем $n/2$ раз. Соответственно, использование предвычислений при умножении на постоянное число позволяет сэкономить примерно $s^2 \cdot n/2$ операций процессорного умножения. Следовательно, применение предложенных предвычислений для ускорения программной реализации модулярного экспоненцирования эффективно при выполнении условия $2^k \cdot s < s^2 \cdot n/2$, которое после преобразований может быть приведено к виду:

$$k \cdot 2^{k+1} < n^2. \qquad (5)$$

Анализ выражения (5) показывает, что для используемых на практике значений n = 1024 и 2048, применение предложенных предвычислений оправдано в случае n = 1024 только при реализации модулярного экспоненцирования на 8-разрядном контроллере, а во втором случае (n = 2048) – при реализации на 8-ми и 16-ти разрядных процессорах.

При постоянном модуле M может быть реализовано дальнейшее ускорение модулярного умножения на постоянное число за счет предвычисления значений

$p_j \cdot M$, что позволит уменьшить количество операций процессорного умножения до s (одно умножение в цикле). Таким образом, будет достигнуто ускорение умножения по сравнению с базовым алгоритмом Монтгомери в $2 \cdot s+1$ раз. При этом, общий объем требуемой табличной памяти для хранения результатов предвычислений составляет $2^{k+1} \cdot (n+k)$ бит. При этом табличная память организована следующим образом: в таблице $T_1(a) = a \cdot B$ объемом 2^k $(n+k)$-разрядных чисел сохранятся значения произведений всех возможных значений k-разрядного кода a на n-разрядный код постоянного числа $B=X$. Младшие k разрядов всех значений этой таблицы представляют собой $T_1'(a) = a \cdot b_0$. Таблица T_1 вычисляется заново перед каждым модулярным экспоненцированием. В таблице $T_2(p)=p \cdot M$ объемом 2^k $(n+k)$-разрядных чисел сохранятся значения произведения всех возможных значений k-разрядного кода p на n-разрядный код постоянного модуля M. Таблица T_2 вычисляется только при изменении ключей алгоритмов защиты информации, которые используют операцию модулярного экспоненцирования.

С учетом введенных изменений и обозначений, модифицированный алгоритм умножения на постоянное число на основе рекурсии Монтгомери – $Mont'(A,B)$, может быть представлен с использованием нотаций C++ в следующем виде:

1. $R = 0$.
2. *for* (j=0; j <= s-1; j++)
   {
   2.1. $p_j = (r_0 + T_1'(a_j)) \cdot M' \bmod 2^k$;
   2.2. $R = (R + T_1(a_j) + T_2(p_j)) >> k$;
   }
3. *if* ($R \geq M$) $R = R - M$.

В цикле предложенного алгоритма используется одна операция процессорного умножения и $2 \cdot s+1$ операций процессорного сложения. Соответственно, при реализации этого алгоритма модулярного умножения требуется s операций процессорного умножения и $2 \cdot s^2+s/2$ операций сложения, в то время, как базовый алгоритм Монтгомери требует $2 \cdot s^2+s$ операций процессорного умножения и $4 \cdot s^2 + 4 \cdot s$ операций процессорного сложения.

Оценка эффективности предложенных алгоритмов

При постоянном модуле M и использовании предвычислений, суммарное количество операций процессорного умножения, необходимых для выполнения модулярного возведения в квадрат составляет $n \cdot s^2/2 + n/2$. Среднее количество операций процессорного умножения, необходимое для умножений на постоянное число в процессе модулярного экспоненцирования составит, в среднем, $s \cdot n/2$. К этому следует добавить количество операций процессорного умножения – $2^k \cdot s$, необходимых для реализации предвычислений, предусмотренных предложенным алгоритмом умножения на постоянное число и реализуемых перед каждым вычислением $X^E \bmod M$. Таким образом, общее число операций процессорного умножения, необходимых для реализации модулярного экспоненцирования при комплексном использовании предлагаемых алгоритмических способов его ускорения составляет, в среднем:

$$N_0 = \frac{n}{2} \cdot (s^2 + s + 1) + 2^k \cdot s. \qquad (6)$$

Сравнительный анализ количества требуемых операций процессорного умножения при обычном модулярном экспоненцировании Монтгомери (1) и с использованием предлагаемых алгоритмов ускоренной реализации модулярного возведения в квадрат и умножения на фиксированное число (6), показывает, что для k=8 и 16 число процессорных операций умножения уменьшается практически в шесть раз, при этом объем памяти, требуемый для хранения результатов предвычислений составляет $2^{k+2} \cdot (n+1)$ бит.

Таким образом, разработанные алгоритмы ускоренного модулярного возведения в квадрат и умножения на постоян-

ное число – двух базовых операций модулярного экспоненцирования, позволяют повысить практически в 6 раз производительность выполнения этой важной для сетевых технологий операции при ее реализации на малоразрядных микропроцессорах, встроенных микроконтроллерах и смарт-картах. Результаты проведенных экспериментальных исследований, в целом, подтверждают приведенные теоретические оценки эффективности предложенных алгоритмов.

Выводы

Производительность реализации модулярного экспоненцирования чисел большой разрядности на малоразрядных микропроцессорах и встроенных микроконтроллерах является одним из важнейших факторов эффективного использования этого класса компьютерных устройств в рамках современных сетевых технологий. Проведенный анализ показал, что существующие алгоритмы модулярного экспоненцирования содержат элементы избыточности и не учитывают особенностей выполнения этой операции в используемых на практике протоколах и алгоритмах защиты информации с открытым ключом.

На основе проведенных исследований предложены модифицированные алгоритмы ускоренного выполнения двух базовых вычислительных процедур модулярного экспоненцирования: модулярного возведения в квадрат и модулярного умножения на постоянное число. Предложенные алгоритмы используют рекурсию Монтгомери, что обеспечивает высокую эффективность ее реализации без использования операций деления. За счет исключения операционной избыточности и использования результатов предвычислений, предложенные алгоритмы позволяют сократить вычислительную сложность реализации модулярного экспоненцирования на малоразрядных микропроцессорах и встроенных микроконтроллерах примерно в шесть раз.

Предложенные алгоритмы ускоренного выполнения вычислительных процедур модулярного экспоненцирования могут быть использованы для повышения производительности программной реализации стандартизированных сетевыми протоколами алгоритмов защиты информации *RSA*, *El-Gamal*, а также алгоритма цифровой подписи *DSA* на малоразрядных встроенных микропроцессорах и микроконтроллерах.

Список литературы

1. *Таненбаум Э.* Компьютерные сети. 4-е изд. – М.: Питер, 2003. – 991 с.

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

3. *Hong S. M., Oh S. Y., Yoon H.* New modular multiplication algorithms for fast modular exponentiation // Proceeding of Advances in Cryptology Eurocrypt'96, LNCS-1070, Springer-Verlag, 1996. – PP. 166-177.

4. *Kawamura S., Takabayashi K., Shimbo A.* A fast modular exponentiation algorithm.// IEICE Transactions., Vol.E-47, № 8, 1991. – PP. 2136-2142.

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

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