КРИПТОЛОГІЯ / CRYPTOLOGY

АЛГОРИТМ ВОЗВЕДЕНИЯ В КВАДРАТ ЦЕЛЫХ ЧИСЕЛ С ИСПОЛЬЗОВАНИЕМ ОТЛОЖЕННОГО ПЕРЕНОСА

Владислав Ковтун,Андрей Охрименко

Национальный авиационный університет, 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 и тезисы докладов на конференциях, патенты и др.
*E-mail:*vladislav.kovtun@gmail.com

ОХРИМЕНКО Андрей Александрович, к.т.н. НАУ

Год и место рождения: 1990, Васильков, Украина.
Образование: Национальный авиационный университет, 2012.
Должность: аспирант кафедры безопасности информационных технологий с 2012 года.
Научные интересы: информационная безопасность, криптосистемы с открытым ключом, сетевая безопасность, оценка рисков.
Публикации: более 40 научных публикаций, включая материалы и тезисы докладов на конференциях, статьи в специализированых научных журналах, авторские свидетельства и др.
E-mail: andrew.okhrimenko@gmail.com

Аннотация. *Авторы предлагают алгоритм возведения в квадрат целых чисел для w-разрядных платформ с использованием подхода к повышению его продуктивности. Предложенный подход основывается на механизме отложенного переноса из старшего бита при накоплении суммы. Эта стратегия позволяет избежать необходимости учета переноса из старшего разряда на каждой итерации цикла накопления суммы. Механизм отложенного переноса позволяет уменьшить общее количество операций суммирования и эффективно применять современные технологии распараллеливания. Теоретические оценки эффективности предложенного алгоритма были подтверждены на практике для 32- и 64-разрядных платформ.*

Ключевые слова: *возведение в квадрат, умножение целых чисел, программная реализация, криптографические преобразования, криптосистема, отложенный перенос.*

1. Введение

Криптографические преобразования с открытым ключом (КПОК) лежат в основе большинства современных криптосистем. Одним из подходов для повышения производительности КПОК, является увеличение быстродействия базовых операций над числами, таких как умножение, возведение в квадрат и приведение по модулю. **Целью** данной работы является поиск путей увеличения производительности КПОК, за счет увеличения быстродействия операций возведения в квадрат целых чисел, используя идею отложенного переноса [1, 3].

Возведение в квадрат является частным случаем умножения, где оба множителя равны [4, 5]. Поэтому, при возведении в квадрат для n-значного числа, необходимо провести только $\left(n^2+n\right)/2$ уникальных операций умножения, в отличие от n^2 операций для умножения.

Используя идею отложенного переноса [1, 3] был предложен алгоритм возведения в квадрат целых чисел. Основой алгоритма возведения в квадрат является предложенный ранее авторами [3] алгоритм умножения ***Modified Comba (MC)***.

2. Алгоритм умножения Modified Comba

Основу алгоритма умножения ***Modified Comba (MC)*** составляет цикл (п. 2 и п. 3), а также вложенный цикл (п. 2.1 и п. 3.1). На низшем уровне иерархии, в цикле п. 2.1, п. 3.1 выполняется умножение и накопление отложенного переноса. Накопленный перенос учитывается на финальных итерациях цикла п. 2 и п. 3. Использование $2w$-разрядных переменных для хранения w-разрядных переменных, позволило избавиться от учета переноса из w-разрядной переменной, после каждой арифметической операции. Перенос накапливается в старшей части 2w-разрядной переменной и может быть учтён при необходимости.

Обобщенный алгоритм ***MC*** [3] для w-разрядных систем приведен ниже:

Алгоритм 1. Умножение целых чисел *MC*

INPUT:

$a, b \in \mathbf{GF}(p)$, $n = \log_{2^w} a$, $nk = 2n-1$

OUTPUT:

$c = a \cdot b$

1. $r_0^{(2w)} \leftarrow 0$, $r_1^{(2w)} \leftarrow 0$, $r_2^{(2w)} \leftarrow 0$.

2. For $k \leftarrow 0$, $k < n$, $k{+}{+}$ do

2.1. For $i \leftarrow 0$, $j \leftarrow k$, $i \le k$, $i{+}{+}$, $j{-}{-}$ do

2.1.1. $(uv)^{(2w)} \leftarrow a_i^{(w)} \cdot b_j^{(w)}$.

2.1.2. $r_0^{(2w)} \leftarrow r_0^{(2w)} + v^{(w)}$, $r_1^{(2w)} \leftarrow r_1^{(2w)} + u^{(w)}$

2.2. $r_1^{(2w)} \leftarrow r_1^{(2w)} + \mathrm{hi}_{(w)}\left(r_0^{(2w)}\right)$, $r_2^{(2w)} \leftarrow r_2^{(2w)} + \mathrm{hi}_{(w)}\left(r_1^{(2w)}\right)$

2.3. $c_k^{(w)} \leftarrow \mathrm{low}_{(w)}\left(r_0^{(2w)}\right)$, $r_0^{(2w)} \leftarrow \mathrm{low}_{(w)}\left(r_1^{(2w)}\right)$,

$r_1^{(2w)} \leftarrow \mathrm{low}_{(w)}\left(r_2^{(2w)}\right)$, $r_2^{(2w)} \leftarrow 0$.

3. For $k \leftarrow n$, $l \leftarrow 1, k < nk$, $k{+}{+}$, $l{+}{+}$ do

3.1. For $i \leftarrow l$, $j \leftarrow k-l$, $i < n$, $i{+}{+}$, $j{-}{-}$ do

3.1.1. $(uv)^{(2w)} \leftarrow a_i^{(w)} \cdot b_j^{(w)}$.

3.1.2. $r_0^{(2w)} \leftarrow r_0^{(2w)} + v^{(w)}$, $r_1^{(2w)} \leftarrow r_1^{(2w)} + u^{(w)}$

3.2. $r_1^{(2w)} \leftarrow r_1^{(2w)} + \mathrm{hi}_{(w)}\left(r_0^{(2w)}\right)$, $r_2^{(2w)} \leftarrow r_2^{(2w)} + \mathrm{hi}_{(2)}\left(r_1^{(2w)}\right)$

3.3. $c_k^{(w)} \leftarrow \mathrm{low}_{(w)}\left(r_0^{(2w)}\right)$, $r_0^{(2w)} \leftarrow \mathrm{low}_{(w)}\left(r_1^{(2w)}\right)$,

$r_1^{(2w)} \leftarrow \mathrm{low}_{(w)}\left(r_2^{(2w)}\right)$, $r_2^{(2w)} \leftarrow 0$.

4. $c_{nk}^{(w)} \leftarrow \mathrm{low}_{(2)}\left(r_0^{(2w)}\right)$.

5. Return (c).

Вычислительная сложность алгоритма ***MC*** представлена ниже:

$$I_{sqr}^{MC} = n^2\left(I_{mul}^{w} + \left(\frac{3n+1}{n}\right)I_{add}^{2w+w}\right),$$

где n – количество w-разрядных машинных слов, необходимых для хранения переменной-множителя заданного размера, I_{mul}^{w} – операция умножения w-разрядных слов, I_{add}^{2w+w} – операция сложения. При расчете вычислительной сложности алгоритмов не учитывается операция присвоения.

3. Алгоритм возведения в квадрат ModifiedComba SQR

Учитывая особенности возведения в квадрат, в алгоритме умножения ***MC*** были внесены изменения во вложенный цикл накопления отложенного переноса (п. 2.1 и п. 3.1) и добавлена дополнительная проверка для исключения повторений при накоплении суммы (п. 2.1.2 и п. 3.1.2).

Ниже представлен алгоритм возведения в квадрат ***Modified Comba SQR (MCSQR)***, для w-разрядных машинных слов, в основе которого лежит алгоритм умножения ***MC*** [1-3].

Вычислительная сложность алгоритма ***MCSQR*** представлена ниже:

$$I_{sqr}^{MCSQR} = n^2\left(I_{mul}^{w} + \left(\frac{3n+1}{n}\right)I_{add}^{2w+w} + 3\left(\frac{n-1}{2n}\right)I_{shift}^{w}\right),$$

где n – количество w-разрядных машинных слов, необходимых для хранения переменной-множителя заданного размера, I_{mul}^{w} – операция умножения w-разрядных слов, I_{add}^{2w+w} – операция сложения, I_{shift}^{w} – операция битового сдвига.

Алгоритм 2. Возведения в квадрат *MCSQR*

INPUT:

$a \in \mathbf{GF}(p)$, $n = \log_{2^w} a$, $nk = 2n-1$

OUTPUT:

$c = a^2$

1. $r_0^{(2w)} \leftarrow 0$, $r_1^{(2w)} \leftarrow 0$, $r_2^{(2w)} \leftarrow 0$.

2. For $k \leftarrow 0$, $k < n$, $k{+}{+}$ do

2.1. For $i \leftarrow 0$, $j \leftarrow k$, $i \le j$, $i{+}{+}$, $j{-}{-}$ do

2.1.1. $\left(u^2\right)^{(2w)} \leftarrow a_i^{(w)} \cdot a_j^{(w)}$.

2.1.2. if $(i < j)$ then $r_0^{(2w)} \leftarrow r_0^{(2w)} + \left(u^{(w)} << 1\right)$,

$r_1^{(2w)} \leftarrow r_1^{(2w)} + \left(\left(u^{(w)} << 1\right) \text{ or } \left(u^{(w)} >> (w-1)\right)\right)$;

else $r_0^{(2w)} \leftarrow r_0^{(2w)} + u^{(w)}$, $r_1^{(2w)} \leftarrow r_1^{(2w)} + u^{(w)}$

2.2. $r_1^{(2w)} \leftarrow r_1^{(2w)} + \mathrm{hi}_{(w)}\left(r_0^{(2w)}\right)$, $r_2^{(2w)} \leftarrow r_2^{(2w)} + \mathrm{hi}_{(w)}\left(r_1^{(2w)}\right)$

2.3. $c_k^{(w)} \leftarrow \mathrm{low}_{(w)}\left(r_0^{(2w)}\right)$, $r_0^{(2w)} \leftarrow \mathrm{low}_{(w)}\left(r_1^{(2w)}\right)$,

$r_1^{(2w)} \leftarrow \mathrm{low}_{(w)}\left(r_2^{(2w)}\right)$, $r_2^{(2w)} \leftarrow 0$.

3. For $k \leftarrow n$, $l \leftarrow 1, k < nk$, $k{+}{+}$, $l{+}{+}$ do

3.1. For $i \leftarrow l$, $j \leftarrow k-l$, $i \le j$, $i{+}{+}$, $j{-}{-}$ do

3.1.1. $\left(u^2\right)^{(2w)} \leftarrow a_i^{(w)} \cdot a_j^{(w)}$.

3.1.2. if $(i < j)$ then $r_0^{(2w)} \leftarrow r_0^{(2w)} + \left(u^{(w)} << 1\right)$,

$r_1^{(2w)} \leftarrow r_1^{(2w)} + \left(\left(u^{(w)} << 1\right) \text{ or } \left(u^{(w)} >> (w-1)\right)\right)$;

else $r_0^{(2w)} \leftarrow r_0^{(2w)} + u^{(w)}$, $r_1^{(2w)} \leftarrow r_1^{(2w)} + u^{(w)}$

3.2. $r_1^{(2w)} \leftarrow r_1^{(2w)} + \mathrm{hi}_{(w)}\left(r_0^{(2w)}\right)$, $r_2^{(2w)} \leftarrow r_2^{(2w)} + \mathrm{hi}_{(2)}\left(r_1^{(2w)}\right)$

3.3. $c_k^{(w)} \leftarrow \mathrm{low}_{(w)}\left(r_0^{(2w)}\right)$, $r_0^{(2w)} \leftarrow \mathrm{low}_{(w)}\left(r_1^{(2w)}\right)$,

$r_1^{(2w)} \leftarrow \mathrm{low}_{(w)}\left(r_2^{(2w)}\right)$, $r_2^{(2w)} \leftarrow 0$.

4. $c_{nk}^{(w)} \leftarrow \text{low}_{(2)}\left(r_0^{(2w)}\right)$.

5. Return (c).

Результаты оценки вычислительной сложности ***MCSQR*** для множителей разной битовой длинны, приведены в табл. 1 и табл. 2 (MUL, ADD и SHIFT – количество необходимых операций умножения, присвоения и сдвига):

Таблица 1

Оценки вычислительной сложности ***MCSQR*** дляw=32 bit

BIT SIZE	MUL	ADD	SHIFT
128	16	52	18
256	64	200	84
512	256	784	360
1024	1024	3104	1488
2048	4096	12352	6048
3072	9216	27744	13680
4096	16384	49280	24384
6144	36864	110784	55008
8192	65536	196864	97920
12288	147456	442752	220608
16384	262144	786944	392448

Таблица 2

Оценки вычислительной сложности ***MCSQR*** для w=64 bit

BIT SIZE	MUL	ADD	SHIFT
128	4	14	3
256	16	52	18
512	64	200	84
1024	256	784	360
2048	1024	3104	1488
3072	2304	6960	3384
4096	4096	12352	6048
6144	9216	27744	13680
8192	16384	49280	24384
12288	36864	110784	55008
16384	65536	196864	97920

Из табл. 1 и табл. 2 можно увидеть, что использование 64-битных машинных слов в алгоритме ***MCSQR*** позволяет существенно сократить количество необходимых операций (в том числе сократить количество операций умножения в 4 раза).

3. Экспериментальное исследование

Алгоритм возведения в квадрат целых чисел ***MCSQR***как и предложенный ранее алгоритм умножения***MC***[1, 3] был программно реализован на языке *C++* с использованием компилятора *Intel C++ Compiler XE 13*. Исследовались реализации алгоритмов для 32- и 64-разрядных платформ. Замеры проводились на компьютере под управлением операционной системы *Microsoft Windows 7 Ultimate x64 SP1* и процессором *Intel Core i5-3570 (6M Cache, 3.40 GHz)* с четырьмя физическими ядрами.

Для умножения двух 64-битных целых чисел использовалась встроенная в компилятор функция *_umul128,* которая позволяет представить128-битный результат умножения в виде массива из двух 64-битных слов.

Сравнение результатов проводилось путем сопоставления среднего времени выполнения умножения в программной реализации алгоритма ***MC*** и предложенного алгоритма возведения в квадрат ***MCSQR***, для 1 млн. итераций, целых чисел размером от 128 до 16384 бит.

Результаты экспериментов для 32-разрядной платформы представлены в табл. 3.

Таблица 3

Результаты экспериментов для w=32 bit

BIT SIZE	MCSQR, ms	MC, ms
128	59	62
256	147	156
512	495	530
1024	1722	1919
2048	6490	7394
3072	14305	16349
4096	24992	28673
6144	54928	63133
8192	97266	110979
12288	214921	246730
16384	378488	435943

Для удобства предлагается нормализовать полученные результаты путем деления временных оценок выполнения ***MC*** на оценки ***MCSQR***. Нормализованные оценки приведены в табл. 4.

Таблица 4

Нормализованные результаты экспериментов для w=32 bit

BIT SIZE	MC/MCSQR
128	1,051
256	1,061
512	1,071
1024	1,114
2048	1,139
3072	1,143
4096	1,147
6144	1,149
8192	1,141
12288	1,148
16384	1,152

Из табл. 4 видно, что предложенный алгоритм возведения в квадрат, при использовании 32-разрядных машинных слов, является эффективнее аналогичных алгоритмов умножения целых чисел. Алгоритма ***MCSQR*** эффективнее алгоритма ***MC*** на 5%, а с ростом битовой длины множителей, выигрыш увеличивается до 15%.

Результаты эксперимента для 64-разрядной платформы, представлены в табл. 5.

Таблица 5

Результаты экспериментов для w=64 bit

BIT SIZE	MCSQR, ms	MC, ms
128	14	15
256	46	50
512	150	163
1024	483	593
2048	1736	2347

Продолжение таблицы 5

3072	3776	5175
4096	6521	8830
6144	14521	19532
8192	25490	34335
12288	56379	76674
16384	99232	135252

Для удобства также предлагается нормализовать полученные результаты путем деления результатов ***MC*** к результатам ***MCSQR***. Нормализованные результаты приведены в табл. 6.

Таблица 6

Нормализованные результаты экспериментов для w=64 bit

BIT SIZE	MC/MCSQR
128	1,071
256	1,087
512	1,087
1024	1,228
2048	1,352
3072	1,370
4096	1,354
6144	1,345
8192	1,347
12288	1,360
16384	1,363

Из табл. 6 видно, что предложенный алгоритм возведения в квадрат, при использовании 64-битных машинных слов, эффективнее аналогичного алгоритма умножения целых чисел. Алгоритма ***MCSQR*** эффективнее алгоритма ***MC*** на 7%, а с ростом битовой длины множителей, выигрыш увеличивается до 36%.

Для анализа эффективности программных реализаций алгоритма возведения в квадрат для 32- и 64-разрядных платформ, предлагается провести сравнение путём деления полученных результатов на 32-разрядной платформе, к результатам, полученным на 64-разрядной платформе. Результаты сравнения представлены в табл. 7.

Таблица 7

Практические оценки алгоритмов

BIT SIZE	MCSQR x86 / MCSQR x64	MC x86 / MC x64
128	4,214	4,133
256	3,196	3,120
512	3,300	3,252
1024	3,565	3,236
2048	3,738	3,150
3072	3,788	3,159
4096	3,833	3,247
6144	3,783	3,232
8192	3,816	3,232
12288	3,812	3,218
16384	3,814	3,223

Из табл. 7 видно, что, как и в случае умножения, программные реализации алгоритмов возведения в квадрат для 64-разрядной платформы оказались до 4 раз эффективнее аналогичной реализации для 32-разрядной платформы. Теоретические оценки сложности алгоритма ***MCSQR*** подтверждаются экспериментальными оценками времени выполнения операций.

4. Выводы

Используя ранее предложенные подходы для повышения быстродействия операции умножения целых чисел [1-3], был разработан алгоритм возведения в квадрат ***Modified Comba SQR***.

Теоретические расчеты показывают, что использование 64-битных машинных слов в алгоритме ***MCSQR***, позволяет сократить количество операций умножения в 4 раза. Программная реализация алгоритма возведения в квадрат ***MCSQR*** для 64-разрядной платформе оказалась эффективнее аналогичной реализации для 32-разрядной платформе до 4 раз. Экспериментальные исследования показали эффективность предложенного алгоритма возведения в квадрат перед алгоритмом умножения, на основании которого он был разработан, как с использованием 32-битных, так и 64-битных слов. Теоретические оценки подтверждаются практическими результатами.

Предложенная идея отложенного переноса позволяет независимо производить сложение результатов соответствующих произведений по столбцам, что дает возможность выполнять накопление суммы старших и младших разрядов в отдельных параллельных потоках. Механизм отложенного переноса позволяет применить несколько подходов к распараллеливанию алгоритма ***MCSQR.*** Дальнейшие исследования будут направлены на применение технологий распараллеливания в алгоритме ***MCSQR.***

Литература

[1] Ковтун В.Ю. Подходы к повышению производительности программной реализации операции умножения в поле целых чисел / Ковтун В.Ю., Охрименко А.А., Нечипорук В.В. // – Защита информации. − №1 (54). – 2012. − С. 68-75.

[2] Ковтун В.Ю. Подходы к распараллеливанию программной реализации операции умножения в поле целых чисел / Ковтун В.Ю., Охрименко А.А. // Радиотехника. Всеукраинский межведомственный научно-технический сборник. — № 171. — Х.: ХНУРЭ, 2012. — С. 123-132.

[3] 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.

[4] Denis T., Rose G. BigNum Math: Implementing Cryptographic Multiple Precision Arithmetic. – Elsevier / Syngress. — 2006. — 336 p.

[5] Hankerson Darrel. Guide to Elliptic Curve Cryptography. / Hankerson Darrel, Menezes Alfred J., Vanstone Scott — Springer-Verlag Professional Computing Series. — 2004. — 311 p.

УДК 004.051/056 (045)

Ковтун В.Ю., Охріменко А.О. Алгоритм піднесення до квадрату цілих чисел з використанням відкладеного переносу

Анотація. *Автори пропонують алгоритм піднесення до квадрату цілих чисел для w-розрядних платформ, з використанням підходу до підвищення його продуктивності. Запропонований підхід ґрунтується на механізмі відкладеного переносу зі старшого біта при накопиченні суми. Ця стратегія дозволяє уникнути необхідності врахування переносу зі старшого розряду на кожній ітерації циклу накопичення суми. Механізм відкладеного переносу дозволяє зменшити загальну кількість операцій суми і ефективно застосовувати сучасні технології розпаралелювання. Теоретичні оцінки ефективності запропонованого алгоритму були підтверджені на практиці для 32- та 64-розрядних платформ.*

Ключові слова: *піднесення до квадрату, множення цілих чисел, програмна реалізація, криптографічні перетворення, криптосистема, відкладений перенос.*

Kovtun V.Yu., Okhrimenko A.O. Integer squaring algorithm with delayed carry mechanism

Abstract. *Authors have offered integer squaring algorithm for w-bit platforms with the approach to increase its performance. This approach relies on delayed carry mechanism of significant bit in sum accumulation. This strategy allows preventing necessity to consider the significant bit carry at the each iteration of the sum accumulation loop. The delayed carry mechanism enables to reduce the total number of additions and apply the modern parallelization technologies effectively. Theoretical estimates of effectiveness of the proposed algorithm have been confirmed in practice for 32 - and 64-bit platforms.*

Key words*: integer squaring, integer multiplication, software implementation, cryptographic transformation, cryptosystem, delayed carry.*

Отримано *24 вересня 2013 року*, затверджено редколегією *16 жовтня 2013 року*