УДК 004.75

Заячук Я.І. к.т.н; **Мойсеєнко О.В.** к.т.н; **Клим'юк М.М.** магістр
(Івано-Франківський національний технічний університет нафти і газу)

ВИКОРИСТАННЯ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ ДЛЯ РЕАЛІЗАЦІЇ АСИМЕТРИЧНИХ КРИПТОАЛГОРИТМІВ НА SIMD-АРХІТЕКТУРАХ НА ПРИКЛАДІ RSA

Заячук Я.І., Мойсеєнко О.В., Клим'юк М.М. Використання системи залишкових класів для реалізації асиметричних криптоалгоритмів на SIMD-архітектурах на прикладі RSA. В статті запропоновано спосіб обчислення RSA шифру на основі системи залишкових класів (RNS), окреслено основні переваги та недоліки запропонованого підходу, наведено приклади обчислень проміжних даних для RSA алгоритму в базисі RNS. Введення доповнених формул для забезпечення відсутності переповнень та зменшення використання пам'яті. Запропоновано новий числовий базис з мінімальною вагою Хеммінга.

Ключові слова: асиметричний криптоалгоритм, SIMD-архітектура, RSA, RNS, система залишкових класів, алгоритм Монтгомері, модульна арифметика

Заячук Я.И., Мойсеенко Е.В., Климьюк Н.М. Применение системы остаточных классов для реализации ассиметричных криптоалгоритмов на SIMD-архитектурах на примере RSA. В статье предложен способ вычисления RSA шифра на основе системы остаточных классов (RNS), обозначены основные преимущества и недостатки предложенного подхода, приведены примеры вычислений промежуточных данных для RSA алгоритма в базисе RNS. Введение дополненных формул для обеспечения отсутствия переполнений и уменьшения использования памяти. Предложен новый численный базис с минимальным весом Хемминга.

Ключевые слова: ассиметричный криптоалгоритм, SIMD-архитектура, RSA, RNS, система остаточных классов, алгоритм Монтгомери, модульная арифметика

Zaiachuk Ia.I., Moiseienko O.V., Klym'iuk M.M. Use of residual classes to implement asymmetric encryption algorithm RSA on SIMD-architectures. This paper proposes a method of calculating RSA cipher based on residue number system (RNS), identified the main advantages and disadvantages of the proposed approach, examples of calculations of intermediate data for the RSA algorithm in the RNS basis. Introduced supplemented formulas to ensure no overflows and reduce memory usage. A new numerical basis with a minimum Hamming weight.

Keywords: asymmetric encryption algorithm, SIMD-architectures, RSA, RNS, residue number system, Montgomery algorithm, modular arithmetic

Вступ. Розробка високопродуктивної система шифрування RSA алгоритмом має велике значення практично для всіх сфер інформаційної безпеки, де особливе значення має час роботи. Це, наприклад, системи видачі цифрових сертифікатів, які повинні опрацьовувати запити від багатьох клієнтів і обчислювати при цьому RSA шифр.

Оскільки RSA шифрування – це практично обчислення результату піднесення до степеня за модулем, а отже є за своєю природою лінійним, потрібно дослідити, як операцію піднесення до степеня за модулем виконати паралельно.

Аналіз операції шифрування/дешифрування в RSA. В алгоритмі RSA використовується операція піднесення до степеня за модулем. Нехай x – вхідне повідомлення, яке потрібно зашифрувати, a – відкритий ключ, d – секретний ключ, n – модуль, який є також частиною ключа. Тоді результат шифрування X буде знаходитись за формулою:

$$X = x^a \ mod \ n \ . \qquad \textbf{(1)}$$

Дешифрування буде здійснюватись аналогічно:

$$x = X^d \ mod \ n \ . \qquad \textbf{(2)}$$

Складність операції спричинена розмірами операндів: 1024 біти та більше. Тому неможливо безпосередньо реалізувати RSA шифрування з адекватною розрядністю ключів

на універсальному процесорі (CPU), який за один такт може здійснювати операції тільки над 64 бітними числами.

Тому виникає потреба у знаходженні способу подання вхідних операндів таким чином, щоб, по-перше, мати змогу зберігати в пам'яті числа великої розрядності; по-друге, процесор міг послідовно опрацьовувати фрагменти цих чисел; по-третє опрацьовані фрагменти могли б бути використані для обчислення виразів (1) та (2), і нарешті – щоб опрацьовування одного фрагменту числа не впливало на опрацьовування інших фрагментів та могло здійснюватись в довільному порядку, що і буде умовою розпаралелюваності.

Базові принципи системи залишкових класів. RNS (від англ. Residue number system – система залишкових класів) – одна з непозиційних систем числення, принципи роботи якої базуються на висновках Китайської теореми про залишки (КТЗ) [1, 2].

В цій системі числа подаються в контексті деякої бази $B=(m_1,m_2,\ldots,m_k)$, яка складається з набору взаємно простих чисел (модулів), де k – кількість елементів бази. При цьому ціле число x з позиційної системи числення в RNS буде представлене набором цілих чисел $(x_1,x_2,\ldots,x_k)$, де $x_i = x \bmod m_i, \quad i=1\ldots k$. КТЗ гарантує однозначну відповідність між числом x та його набором залишків, якщо $0 \le x < M$, де $M=\prod_{i=1}^{k} m_i$. На основі доведення КТЗ можна отримати формулу для зворотнього перетворення числа, тобто з RNS в позиційну систему числення:

$$x = \sum_{i=1}^{k} x_i M_i \left| M_i^{-1} \right|_{m_i} \bmod M, \qquad \textbf{(3)}$$

де $M_i = \dfrac{M}{m_i}$ і $\left|M_i^{-1}\right|_{m_i}$ є інверсією M_i за модулем m_i. Надалі у формулах замість ($x$mod$m$) буде записуватись $|x|_m$.

Якщо числа x та y подані в RNS формах $(x_1,x_2,\ldots,x_k)$ та $(y_1,y_2,\ldots,y_k)$ відповідно, то отримаємо формули для операцій додавання, віднімання та множення:

$$x \pm y = (|x_1 \pm y_1|_{m_1}, \ldots, |x_k \pm y_k|_{m_k}). \qquad \textbf{(4)}$$

$$x \times y = (|x_1 \times y_1|_{m_1}, \ldots, |x_k \times y_k|_{m_k}). \qquad \textbf{(5)}$$

Експоненціювання за модулем в RNS базується на множенні Монтгомері, яке повторюється до отримання результату. Подання числа x в базисі Монтгомері виглядає як знаходження значення виразу $xR^{-1} \bmod N$. Для цього шукають таке $q < R$, що $x + qN$ є кратним до R. Тут $R>N$, НСД(R,N) = 1 та $0 \le x < RN$.

Отже знаходження значення виразу $y=(x+qN)R^{-1}$ відбувається без операції ділення. Кінцевий результат $x \bmod N$ отримують, використовуючи цей же алгоритм з вхідними даними y та (R^2mod$N$).

Аналогічно алгоритм Монтгомері знаходить значення виразу $xyR^{-1} \bmod N$. Для практичності як константу Монтгомері R обирають число, яке є степенем двійки, щоб замінити операцію множення на R^{-1} простими бітовими зсувами.

І, нарешті, результат піднесення до степеня знаходять, рекурсивно обчислюючи значення виразу $xyR^{-1} \bmod N$, де, в одному випадку, x та y є одним і тим же числом, тобто шукається квадрат числа за модулем, а в іншому – x є результатом обчислення квадрату

попередньої дії, а *y* – початковим числом, степінь якого шукається. В даному випадку застосовується модифікація алгоритму.

Таким чином, головною складністю знаходження результату експоненціювання за модулем в системі RNS є виконання множення Монтгомері, яке, на жаль, володіє певною складністю, зокрема через те, що вимагає для свого виконання використання двох RNS базисів, оскільки інверсія *M* за модулем більша, ніж значення цього модуля. Допоміжний базис такий, що $НСД(m_i', M) = 1$ для $i = 1 \ldots l$, $M' = \prod_i m_i'$ і $M' \geq M$.

Реалізація алгоритму Монтгомері в RNS. Загалом алгоритм Монтгомері *MM(a, b, N)*, який можна буде застосовувати для чисел в RNS, складається з такої послідовності кроків[3]:

Вхід: – Два RNS базиси $B = (m_1, \ldots, m_k)$ та $B' = (m_{k+1}', \ldots, m_{2k}')$ такі, що $M = \prod_{i=1}^{k} m_i < M' = \prod_{i=1}^{k} m_{k+i}$ та $НСД(M, M') = 1$.

– Надлишковий модуль (m_r, $НСД(m_r, m_i) = 1 \forall i = 1 \ldots 2k$) = 1.

– Натуральне число *N*, подане в двох *RNS* базисах $0 < (k+2)^2 N < M$ та $НСД(N, M) = 1$.

– Два натуральних множники *a, b*, подані в двох *RNS* базисах, з умовою *ab<MN.*

Вихід: натуральне число $\hat{r} \equiv abM^{-1} (\mathrm{mod} N)$ в RNS подане для двох базисів, при чому $\hat{r} < (k+2)N$.

Послідовність кроків виглядає так:

1. $q \leftarrow (a \times b) \times (-N^{-1})$ в базисі *B*.
2. $[q \, в \, B] \rightarrow [\hat{q} \, в \, B']$ – перше розширення базису (пряме).
3. $\hat{r} \leftarrow (a \times b + \hat{q} \times N) \times M^{-1}$ в базисі B'.
4. $[\hat{r} \, в \, B] \leftarrow [\hat{r} \, в \, B']$ – друге розширення базису (зворотне).

Кроки 1 та 3 складаються винятково з модулярних RNS операцій і можуть бути виконані паралельно. Вся обчислювальна складність алгоритму припадає на два розширення базису – немодулярної операції. Загалом алгоритм подібний до алгоритму, запропонованого у працях Posch і Posch [1] та Kawamura [4], де також здійснюється два розширення бази. В 2002 році Laurent Imbert та Jean-Claude Bajard запропонували новий ефективний спосіб розширення базису, який вимагає менше елементарних операцій на відміну від [5] та [6].

Перше (пряме) розширення базису. Оскільки це другий крок алгоритму Монтгомері, на даному етапі відомо значення *q* для базису *B*. Якщо обчислити суму, спочатку обчисливши проміжний масив коефіцієнтів:

$$\sigma_i = q_i \mid M_i^{(-1)} \mid_{(m_i)} mod m_i, \qquad \textbf{(6)}$$

отримаємо $q = \sum_{i=1}^{k} M_i \sigma_i - \alpha M$, де α – деяке ціле додатнє число, менше за *k*.

J.-C. Bajard [7] запропонував обчислювати не точне значення $\hat{q}$ в базисі B', а зміщене на константу αM, внаслідок чого відпадає потреба в обчисленні коефіцієнта α:

$$\widehat{q_j} = \left| \sum_{i=1}^{k} \left| M_i \right|_{m_j} \sigma_i \right|_{m_j}, \ \forall j = k+1..2k. \qquad \textbf{(7)}$$

Третій крок цілком тривіальний – на основі відомого значення $\hat{q}$ в базисі B', знаходиться значення $\hat{r}$ в тому ж базисі: $\hat{r} = (ab + \hat{q}N)M^{-1} = (ab + qN)M^{-1} + \alpha N < M'$.

Результат, отриманий до цього моменту цілком задовільний, однак потрібно зважити на те, що алгоритм Монтгомері буде виконуватись на потокових процесорах відео карти Nvidia, де максимальне значення змінної може бути рівним 2^{64}–1=18446744073709551615 для змінної типу *unsignedlonglong*, а цього діапазону чисел може за певних умов виявитись недостатньо, що спричинить виникнення переповнень і неправильного результату.

Тому запропоновано виконати додаткові ділення за модулем, щоб не допустити виникнення переповнення:

$$\sigma_i = \left| \left| q_i \right|_{m_i} \times \left| M_i^{-1} \right|_{m_i} \right|_{m_i} . \qquad \textbf{(8)}$$

Формулу (8) перетворимо використовуючи (4) та (5):

$$\hat{q_j} = \left| \sum_{i=1}^{k} \left| \left| M_i \right|_{m_j} \times \left| \sigma_i \right|_{m_j} \right|_{m_j} \right|_{m_j}, \quad \forall j = k+1..2k . \qquad \textbf{(9)}$$

Отримані формули відрізняються від попередніх (6) та (7) більшою кількістю ділень за модулем, це незначно ускладнює обчислення результату, проте гарантує убезпечення від переповнень.

Друге (зворотне) розширення базису. Скористаємось способом, запропонованим в роботі [7]. Спочатку обчислимо значення масиву γ :

$$\gamma_j = \hat{r}_j \,|\, M_j^{(\prime} - 1) \,|_{(m_j)} \; mod m_j, \quad \forall j = k+1...2k \; . \qquad \textbf{(10)}$$

В базисі B' значення масиву $\hat{r}$ буде визначатись так:

$$\hat{r} = \sum_{j=1}^{k} M'_j \gamma_j - \beta M', \beta < k.$$

Отже, зворотне розширення в базис B можна здійснити, знаючи значення константи β :

$$\left| \hat{r} \right|_{m_i} = \left| \sum_{j=1}^{k} \left| M'_j \right|_{m_i} \gamma_j - \left| \beta M' \right|_{m_i} \right|_{m_i}, \forall i = 1 \ldots k \; . \qquad \textbf{(11)}$$

Знаходження константи β є також досить трудомістким:

$$\beta = \left| \left| M'^{-1} \right|_{m_r} \left(\sum_{j=1}^{k} \left| M'_j \right|_{m_r} \gamma_j - \left| \hat{r} \right|_{m_r} \right) \right|_{m_r} . \qquad \textbf{(12)}$$

Всі операції здійснюються за модулем m_r, тому логічно вибрати його таким, щоб операція ділення виконувалось максимально просто – у випаду, якщо $m_r = 2^n$, то результат $\left| x \right|_{mr}$ буде рівний *n* першим бітам числа *x*.

Зазначимо також, що значення $\left| M'^{-1} \right|_{m_r}$, $\left| M'_j \right|_{m_r}$ можна обчислити заздалегідь на центральному процесорі і зберегти в пам’яті. Пізніше, коли буде потрібно, просто зчитати потрібне значення з пам’яті.

Розглянемо детальніше формулу (12). Враховуючи, що максимальне значення γ_j *та* $\left| M'_j \right|_{m_i}$ може бути в межах 2^{32} – 1, максимальне теоретичне значення їхнього добутку буде (2^{32} – 1)(2^{32} – 1) = 0xFFFFFFFE00000001, тож їхня сума, досить імовірно, може не

вміщуватись в межі [0, $2^{64} - 1$]. В цьому випадку залишається вдатись до неприємного, та, на жаль, неминучого кроку, а саме – використовувати для збереження вказаної суми у двох сусідніх змінних типу *unsignedlonglong*, що дасть змогу зберігати числа з діапазону [0, $2^{128} - 1$], і з ускладненням арифметичних операцій, які в даному випадку потрібно буде виконувати над двома операндами, замість одного.

Модифікація модулярних кроків алгоритму Монтгомері. Подібно, як при розширеннях базису, в першому та третьому кроках також можуть виникати переповнення.

Розглянемо крок 1 алгоритму Монтгомері:

$$\left|q\right|_{m_i} \leftarrow \left|(a \times b) \times (-N^{-1})\right|_{m_i}.$$

Кожний з трьох множників *a, b* чи $-N^{-1}$, може бути як завгодно близьким до значення $2^{32} - 1$, при чому одночасно. В такому випадку 64 біт не буде достатньо для збереження добутку. Крім цього, навіть застосувавши формулу (10), є імовірність виникнення переповнення у випадку обрання базису, що складається з достатньо великих модулів. Зважаючи на це, пропонується розділити знаходження кінцевого результату на декілька етапів, на кожному з яких відбувалося б ділення за модулем поточного добутку. Такі дії не суперечать (3), оскільки:

$$\left|q\right|_{m_i} \leftarrow \left|(a \times b) \times (-N^{-1})\right|_{m_i} \equiv \left|\left|a\right|_{m_i} \times \left|b\right|_{m_i} \times \left|-N^{-1}\right|_{m_i}\right|_{m_i} \equiv \left|\left|\left|a\right|_{m_i} \times \left|b\right|_{m_i}\right|_{m_i} \times \left|-N^{-1}\right|_{m_i}\right|_{m_i}.$$

При такому підході в пам'яті комп'ютера на кожному кроці потрібно буде зберігати результат множення двох чисел, які менші за $2^{32} - 1$, тобто менше за $(2^{32} - 1) \times (2^{32} - 1)$ і переповнення не виникне.

Розглянемо крок 3 алгоритму Монгомері. Він забезпечує отримання результату в розширеному базисі B':

$$\hat{r} \leftarrow (a \times b + \hat{q} \times N) \times M^{-1}.$$

Міркування щодо модифікації будуть аналогічними до тих, які були здійснені до кроку 1. Проте в цьому випадку переповнення більш імовірні, оскільки результат $a \times b + \hat{q} \times N$ може не поміщатись в пам'яті. Формулу для обчислення $\hat{r}$ можна подати у вигляді:

$$\left|\hat{r}\right|_{m_j} \leftarrow \left|(a \times b + \hat{q} \times N) \times M^{-1}\right|_{m_j} \equiv \left|\left|a \times b + \hat{q} \times N\right|_{m_j} \times \left|M^{-1}\right|_{m_j}\right|_{m_j} \equiv \left|\left|\left|a \times b\right|_{m_j} + \left|\hat{q} \times N\right|_{m_j}\right|_{m_j} \times \left|M^{-1}\right|_{m_j}\right|_{m_j}.$$

Послідовність кроків, на які можна розбити виконання обчислення $\left|\hat{r}\right|_{m_j}$, щоб уникнути переповнення *unsignedint*, виглядатиме так:

1. $\left|\hat{r}\right|_{m_j} \leftarrow \left|\left|a \times b\right|_{m_j} + \left|\hat{q} \times N\right|_{m_j}\right|_{m_j}$.

2. $\left|\hat{r}\right|_{m_j} \leftarrow \left|\left|\hat{r}\right|_{m_j} \times \left|M^{-1}\right|_{m_j}\right|_{m_j}$.

Очевидно, що для знаходження результату в системі залишкових класів будуть використовуватись проміжні суми, які знаходяться на кроці 3 алгоритму Монтгомері.

Обчислення залишку від ділення на надлишковий модуль. Це обчислення є досить нетривіальною задачею, якщо ставиться завдання отримати високу швидкодію цієї операції. І більшість авторів мало звертає увагу на способи обчислення $\left|x\right|_{mr}$, якщо *x* задане в RNS.

Простий спосіб передбачає зворотну конвертацію числа *x* у ПСЧ на основі формули

$$|x|_{m_r} = \left| \sum_{i=1}^{k} x_i M_i \left| M_i^{-1} \right|_{m_i} \right|_M \Bigg|_{m_r}, \qquad \textbf{(13)}$$

і взяття *n* молодших бітів ($2^n = m_r$). Складність полягає у необхідності зберігання чисел великої розрядності та виконання над ними операцій додавання, множення та ділення за модулем. При такому підході більшість ресурсів відеокарти буде витрачено на обчислення $|x|_{mr}$, яке необхідно виконувати кожен раз для інших вхідних даних. Співвідношення (13) не рекомендується застосовувати на практиці через те, що воно вимагає багато обчислень і є неефективним. Для 1024-бітного ключа значення *M* буде дещо більше, ніж 1024 біти завдовжки, з урахуванням надлишковості, воно буде добутком 33 псевдопростих чисел Мерсенна [8], а зберегти таке велике число, а також всі проміжні суми добутків, досить проблематично, враховуючи контекст виконання програми.

В роботі J. Pollard [7] запропонований дуже цікавий спосіб вирішення цієї проблеми, і він базується на використанні змішаної системи залишків (MRS), яка є узагальненням ПСЧ.

Як і для RNS, визначається MRS базис з набору цілих додатних чисел ($m_1, m_2, ..., m_n$) та ціле число X, подане як ($x_1^{'}...x_n^{'}$) з умовою $x_i^{'} < m_i$ для $i \in \{1,\ldots,n\}$ і воно обчислюється за формулою:

$$X = x_1^{'} + x_2^{'} m_1 + x_3^{'} m_1 m_2 + \ldots + x_n^{'} m_1 \ldots m_{n-1}. \qquad \textbf{(14)}$$

Обчислення $x_i^{'}$ здійснюється рекурсивно, починаючи з першого елемента:

$$\begin{cases} x_1^{'} = |x_1|_{m_1}, \\ x_2^{'} = \left| \left(x_2 - x_1^{'} \right) m_{1,2}^{-1} \right|_{m_2}, \\ x_3^{'} = \left| \left(\left(x_3 - x_1^{'} \right) m_{1,3}^{-1} - x_2^{'} \right) m_{2,3}^{-1} \right|_{m_3}, \\ . \\ x_n^{'} = \left| \left(\ldots \left(x_n - x_1^{'} \right) m_{1,n}^{-1} \ldots - x_{n-1}^{'} \right) m_{n-1,n}^{-1} \right|_{m_n}. \end{cases} \qquad \textbf{(15)}$$

Таким чином, знаючи значення всіх $x_i^{'}$, тобто MRS подання числа X, можна застосувати ділення за модулем для кожного з доданків:

$$\widetilde{|X|}_{m_r} = \left| x_1^{'} + m_1 (x_2^{'} + m_2 (x_3^{'} + \ldots + m_{n-1} x_n^{'}) \ldots) \right|_{m_r}. \qquad \textbf{(16)}$$

Порівняємо складності обох способів обчислення остачі від ділення на надлишковий спосіб. Обчислення підмодульної суми вимагає *k* додавань, кожний доданок – це 2 множення (вважаємо, що M_i *та* $\left|M_i^{-1}\right|_{m_i}$ обчислені заздалегідь, і на цьому етапі просто зчитуються з пам'яті). Отож маємо 2*l* арифметичних операцій. Кількість операцій визначатиметься так:

$$N(\text{“+”}) = (m+\alpha)\times(4n),\ N(\text{“–”}) = \beta \times n \times m,\ N(\text{“<>=”}) = 2 \times n \times r \times m,$$
$$N(\text{“>>”}) = k + 1,\ N(\text{“<<”}) = m \times n \times \alpha + 2 \times r \times m \times n,$$

де *n* – кількість фрагментів, з яких складається одне велике число; *m* – кількість потоків (threads); α – вага Хемінга одного з операндів; β – кількість додатних результатів порівняння в при знаходженні $|X|_M$; *r* – бітова довжина змінної типу unsignedint (зазвичай *r* = 32);

$k = \log_2 m_r$; (“>>[1]”) – операція “елементарний зсув вправо на 1”, (“<>=”) – операція порівняння двох довгих чисел, (“>>”) та (“<<”) – зсув довгого числа на 1 біт відповідно вправо і вліво, (“+”),(“–”) – додавання та віднімання довгих чисел.

Величина *N(“+”)* – кількість елементарних операцій – є не сталою і коливається від $m \times (4n)$ до $(m+\alpha) \times (4n)$. Коефіцієнт α дорівнює вазі Хеммінга першого множника. Також не сталою є кількість елементарних зсувів. Для простоти вважатимемо, що кількість всіх операцій максимальна. Також знехтуємо часом на обнулення початкових масивів, копіювання елементів з одного масиву в інший (хоча при великій кількості повторень вказані операції будуть суттєво сповільнювати процес обчислень).

Отже можна отримати таку характеристичну формулу для підрахунку кількості елементарних операцій у вказаному алгоритмі:

$$N = 9n \times 10m \times 4r + \sigma \times \alpha\beta + k. \quad \textbf{(17)}$$

Тут σ – деяке ціле число, яке залежить від характеру вхідних операндів. Для 1024-бітного ключа, значення змінних *n* та *m* буде відповідно 32 та 33, тому можна говорити про сумарну кількість елементарних операцій, порядок її – 10^6.

Розглянемо алгоритм на основі MRS подання. За формулами арифметичної прогресії знайдемо сумарну кількість потрібних операцій:

$$N("-") = N("\times") = \sum_{i=1}^{n} a_i = \frac{1+n}{2} n, \quad N("mod") = n. \quad \textbf{(18)}$$

Безпосереднє ж обчислення $\left|\widetilde{X}\right|_{m_r}$ можна здійснити такою ж кількістю операцій, з єдиною відмінністю – тут замість операції “віднімання” буде операція “додавання”:

$$N("+") = N("\times") = \sum_{i=1}^{n} a_i = \frac{1+n}{2} n, \quad N("mod") = n \ . \quad \textbf{(19)}$$

Отримаємо таку формулу для визначення кількості потрібних операцій:

$$N = (1+m)m + \frac{1+s}{2} s + \frac{1+a}{2} a + 2d, \quad \textbf{(20)}$$

де *m* – кількість елементарних операцій множення; *s* – кількість елементарних операцій віднімання; *a* – кількість елементарних операцій додавання; *d* – кількість елементарних операцій ділення за модулем.

Порівнюючи (17) та (20) можна зробити декілька висновків.

По-перше, не потрібно витрачати пам’ять та обчислювальні ресурси на збереження довгих чисел – весь процес рекурсивно відбувається над коефіцієнтами числа, поданого в MRS.

По-друге, відсутні складні операції бітових зсувів, реалізація яких вимагає обчислення бітів переносу. І нарешті, останній, але чи не найважливіший висновок – кількість елементарних операцій, яка для цього потрібна – вона не залежить від кількості потоків (threads), чи фрагментів довгого числа, а тільки від *n* – кількості модулів в базисі MRS.

Обґрунтування вибору RNS базисів. Правильний вибір набору модулів, за якими шукається остача від ділення, досить непросте завдання. Потрібно шукати компроміс між можливістю обчислити результат за меншу кількість кроків та доступною кількістю простих модулів, які знаходяться в межах від 0 до $2^{32} - 1$, тобто один модуль можна зберегти в *unsignedint***.**

Найбільш оптимальним є використання як модулів *простих чисел Мерсенна* [9]. Це числа вигляду $m = 2^k - 1$. Враховуючи властивість $\left|2^k\right|_m = 1$, операцію "ділення за модулем" можна значно спростити і замінити однією операцією додавання або однією операцією додавання та однією операцією віднімання. Якщо подати x у вигляді $x = x_1 \times 2^k + x_0$, то отримаємо:

$$\left|x\right|_m = \begin{cases} x_1 + x_2, \; x_1 + x_2 < m, \\ x_1 + x_2 - m, \; x_1 + x_2 \geq m. \end{cases}$$

Проте, цей підхід непрактичний, а все через те, що існує тільки 1 число Мерсенна, яке складається з k біт, крім цього обрання таких чисел для бази RNS спричинить неконсистентність числового поля, будуть присутні числа різного порядку, що може ускладнити аналіз проміжних результатів.

Псевдопрості числа Мерсенна. Ця множина чисел є розширенням класу чисел Мерсенна, яких було недостатньо для практичних цілей [9]. Це числа вигляду:

$$m = 2^k - c, k \in \mathbb{N}, c < 2^{\frac{k}{2}}.$$

Операція ділення за модулем буде вимагати (2×H(c) + 2×k)-бітні додавання, де *H(c)* – Хеммінгова вага *c.* Отже рекомендовано обирати такі *c*, щоб у них було мінімальне значення ваги Хеммінга.

Отже, операція RSA шифрування може бути зведена до багаторазового повторення операції добутку Монтґомері над RNS компонентами, і до того ж незалежно для кожної ітерації. Кількість ітерацій залежить від ваги Хеммінга, тому для публічної експоненти зазвичай обирають невеликі числа за модулем і з низькою вагою (найчастіше – це 65537_{10} = 0x10001). Отриманий метод дає змогу виконати шифрування RSA на пристроях з SIMD-архітектурою (елементи цієї архітектури містять більшість сучасних універсальних процесорів; повністю її реалізують більш спеціалізовані – відеопроцесори), і за прийнятний час шифрувати повідомлення з 2048-бітними ключами і більше.

Література

1. Modular arithmetic [Електронний ресурс] // – Режим доступу: http://en.wikipedia.org/wiki/Modular_arithmetic.

2. Knuth D. The Art of Computer Programming, Volume 2: Semi numerical Algorithms, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89684-2.

3. Sandifer C. The early mathematics of Leon hard Euler. – MAA, 2007. – C. 391. – ISBN 0-88385-559-3.

4. Sandee M. RSA-512 Certificates abused in the wild. [Електронний ресурс] // – Режим доступу: http://blog.fox-it.com/2011/11/21/rsa-512-certificates-abused-in-the-wild/.

5. Simon S. The Evolution Of Secret Writing / The Code Book – The Secret History of Codes& Code-breaking (англ.). – London: Forth Estate, 2000. – P. 3-14. – ISBN 1-85702-889-9.

6. David K. Heterogeneous Impulses / The Codebreakers – The Story of Secret Writing. – NewYork : Charles Scribner's Sons, 1967. – 473 c. – ISBN 0-684-83130-9.

7. Simon S. The Arab Cryptanalysts / The Code Book – The Secret History of Codes& Code-breaking (англ.). – London: ForthEstate, 2000. – P. 14-20. – ISBN 1-85702-889-9.

8. Montgomery P. L. Modular multiplication without trial division. Mathematics of Computation, 44(170):519–521, April 1985. – ISBN 0-3562-55636-8.

9. Robert D. Silverman. Has the RSA algorithm been compromised as a result of Bernstein's Paper? [Електронний ресурс] //– Режим доступу: http://www.rsa.com/rsalabs/node.asp?id=2007.