УДК 004.056.55 **DOI:10.30837/rt.2020.1.200.02**

О.Г. КАЧКО, канд. техн. наук, Ю.І. ГОРБЕНКО, канд. техн. наук, В.А. ПОНОМАР, канд. техн. наук, М.В. ЄСІНА, канд. техн. наук, С.О. КАНДІЙ

ОПТИМІЗАЦІЯ АЛГОРИТМУ МНОЖЕННЯ ПОЛІНОМІВ ДЛЯ NTRU-ПОДІБНИХ АЛГОРИТМІВ

Вступ

Наразі актуальною стала проблема криптографічного захисту від класичних та потенційних криптоаналітичних атак з використанням квантового комп'ютера та квантової математики. Розуміючи цю проблему, технологічно розвинені держави направляють суттєві зусилля на аналіз криптографічної стійкості існуючих стандартів криптографічного захисту інформації у постквантовий період та ведуть пошук щодо створення постквантових стандартів асиметричної криптографії. Практичне вирішення цієї проблеми здійснюється на світовому рівні в процесі проведення NIST США міжнародного конкурсу [1]. Причому особливу увагу звернуто на створення Асиметричного шифру (АСШ) та протоколу інкапсуляції ключів (ПІК). Завершено перший раунд (етап) та йде другий етап. У ході другого етапу проведено проміжний семінар, за рішенням якого NIST США рекомендував до подальших досліджень 17 криптопримітивів сумісної реалізації АСШ та ПІК [1 – 3].

Як показали попередні дослідження, надійною математичною основою, на якій можуть бути створені постквантові АСШ та ПІК, нині вважаються алгебраїчні решітки. Спираючись на вказане та виконані національні дослідження, в Україні вже у 2019 р. прийнято національний стандарт ДСТУ 8961:2019 «Алгоритми асиметричного шифрування та інкапсуляція ключів». В основу його побудови якраз і покладено алгебраїчні решітки. Асиметричне шифрування та інкапсуляцію ключів в стандарті виконують на основі математичних перетворень в кільці поліномів над скінченим полем. Також у стандарті враховані вимоги щодо забезпечення криптографічної стійкості проти спеціальних атак на основі витоку по технічних каналах, а також потенційних класичних та квантових атак. Стандарт розроблено з урахуванням досвіду створення та застосування ДСТУ ISO/IEC 18033-2:2015 тощо [3].

Стандарт, залежно від рівня криптографічної стійкості проти класичних та квантових атак, яку необхідно забезпечити, можна застосовувати в трьох режимах роботи АСШ та ПІК:

- режим СКЕЛЯ–КЕМ 256/128 – 256 біт захисту від класичних атак та 128 біт захисту від квантових атак, а також захисту від спеціальних атак;
- режим СКЕЛЯ–КЕМ 384/192 – 384 біт захисту від класичних атак та 192 біт захисту від квантових атак, а також захисту від спеціальних атак;
- режим СКЕЛЯ–КЕМ 512/256 – 512 біт захисту від класичних атак та 256 біт захисту від квантових атак, а також захисту від спеціальних атак.

Також в кожному із режимів роботи можна застосовувати окремо такі криптографічні перетворення:

- незалежний алгоритм (функція) асиметричного шифрування;
- протокол інкапсуляції ключів (функція), що ґрунтується на застосуванні функції асиметричного шифру;
- механізм (функція) симетричного шифрування та автентифікації, що ґрунтується на функціях асиметричного шифрування та інкапсуляції ключів.

У кожному із режимів роботи внаслідок застосування криптографічних перетворень в кільцях поліномів та скінчених полях забезпечують надання послуг конфіденційності, цілісності, справжності, доступності та криптографічної живучості ключа сеансу зв'язку та його узгодження між відправником та отримувачем.

Попередній аналіз показав, що для забезпечення 5 – 7 рівнів криптографічної стійкості необхідно суттєво збільшувати розміри (довжини) параметрів та ключів вказаних алгоритмів криптографічних перетворень АСШ та ПІК на решітках. Але при збільшенні розмірів пара-

метрів та ключів виникла проблема зменшення часової складності алгоритмів генерування параметрів та ключів, а також прямих та зворотних криптографічних перетворень АСШ та ПІК.

Метою статі є оптимізація алгоритму множення поліномів за критерієм часової складності, який використовується для генерування ключів та виконання прямих та зворотних криптографічних перетворень АСШ та ПІК на алгебраїчних решітках.

1. Постановка та аналіз задач досліджень

В NTRU-подібних алгоритмах асиметричних криптоперетворень [4, 5, 7] основними складовими є алгоритми генерування ключів та виконання прямих та зворотних криптографічних перетворень. Так, наприклад, в NTRU-подібних алгоритмах відкритий ключ h, це поліном розміром n (n – просте число), який обчислюється за особистим (F, G) згідно (1), або подібної формули [4]

$$h = G * F^{-1}, \tag{1}$$

де G, F – поліноми розміром n з коефіцієнтами за малим модулем p (зазвичай, p=3). Обчислення виконуються за великим модулем q, який залежить від алгоритму і може бути ступенем 2 або простим числом. В якості прикладів можна назвати модулі з довжиною q=2048, 4591 біт тощо.

Розглянемо, які алгоритми потрібно оптимізувати. Для визначення найбільш обчислювально складної операції реалізації NTRU-подібних алгоритмів розглянемо алгоритми генерації ключів, зашифрування та розшифрування для [4].

Генерація ключів:

Дано: F (1, -1, 0), G (1, -1, 0). Кількість ненульових елементів в поліномах фіксована і залежить від алгоритму.

Обчислити:

$$f = pF + 1; \quad h = pGf^{-1} (\mathrm{mod}\, q) \tag{2}$$

Зашифрування.

Дано: Повідомлення m, відкритий ключ h, випадковий компонент *seed*.

Обчислити:

$$M = m \,||\, seed; \quad r = \lambda(M); \tag{3}$$

$$m' = M \wedge H(r * h (\mathrm{mod}\, q)); \tag{4}$$

$$E = (r * h + m') \bmod q. \tag{5}$$

Розшифрування.

$$m' = ((f * E) \bmod q) \bmod p; \tag{6}$$

$$r * h \bmod q = E - m'; \tag{7}$$

$$M = m' \wedge H(r * h \bmod q); \qquad r = \lambda(M); \tag{8}$$

Якщо E=(r*h+m')mod q, то розшифрування вірне, інакше – помилка.

Усі операції виконуються в кільці поліномів чи в кільці поліномів над скінченим полем, що визначається алгоритмом.

Аналіз алгоритмів генерації ключів, зашифрування та розшифрування показує, що найбільш складною є операція множення поліномів в скінчених полях [6]. Причому, для запобігання атак сторонніми каналами необхідно забезпечити незалежність часу виконання операцій множення поліномів від розміщення ненульових елементів ключа. У [5, 6, 8] для множення поліномів пропонується використовувати метод Тоома разом з методом Карацуби [9],

в [7, 10] – метод Number TheoreticTransform (NTT), але їх використання зменшує продуктивність та накладає свої обмеження.

Мета та методи оптимізації

Оскільки основу алгоритмів генерації ключів, зашифрування та розшифрування складає операція множення поліномів, то в подальшому розглядаються методи та результати оптимізації алгоритму множення поліномів в кільцях поліномів над скінченими полями. Мета оптимізації – досягнення найменшої обчислювальної складності в порівнянні з уже досягнутими результатами [7, 8, 12, 17] за умови неможливості використання часу виконання функції множення для отримання додаткової інформації про особистий ключ. При оптимізації будемо враховувати не тільки алгоритмічні можливості, які дозволяють зменшити кількість операцій, а і засоби оптимізації, які враховують структуру сучасних процесорів, а саме – мінімізацію помилок в прогнозуванні переходів, кількості «промахів» значень кешу, звертання до не «вирівняної» пам'яті [13] тощо.

2. Сутність та властивості алгоритмів оптимізації

Існуючі алгоритми множення поліномів поділимо на 2 класи. До першого класу віднесемо алгоритми, які використовують властивості структури особистого ключа. До другого – алгоритми, аналогічні швидкому перетворенню Фур'є тощо. В першому випадку вдається врахувати, що більшість коефіцієнтів поліному, що визначає особистий ключ NTRU, дорівнюють 0, а ненульові коефіцієнти дорівнюють 1 або -1. Тому операцію множення можна замінити операціями додавання та віднімання, що дає хороші шанси на зменшення часу обчислення. В другому випадку використовуються можливості оптимізації засобом використання методу Тоома – Кука, теоретико-числових перетворень, швидких перетворень Фур'є [9, 11, 14], тощо.

Що стосується алгоритмів другого класу, то використання методу Тоома – Кука разом з методом Карацуби розглянуто в [5, 12]. Цей метод найбільш ефективний, якщо залишок від ділення N на 2^k близький до 2^k. Саме такий параметр обрано розробниками NTRU Prime [2] (N=761, 761 mod 128 = 121). Використання теоретико-числових перетворень (Number TheoreticTransform – NTT) [7, 10] має суттєве обмеження щодо використання, а саме вимоги $q \equiv 1 (\mathrm{mod} 2*N)$ і q – просте число. Саме такі параметри обрані розробниками CRYSTALS-DILITHIUM [2, 7]. Там же досліджено ефективність цього методу. Наші дослідження показали, що використання швидкого перетворення Фур'є [14] значно поступається продуктивності іншим методам, тому в статті результати не наводяться.

Методи, що пропонуються, відносяться до першого класу алгоритмів, їх можна використовувати для довільних значень q, N без обмежень. Основною метою їх використання є оптимізація операцій множення поліномів за умови виключення або мінімізації можливості отримання інформації про структуру ключа.

Опис запропонованих алгоритмів

Нехай поліном a складається з N елементів, кожний елемент за модулем q, значення елементів в інтервалі $[0..q–1]$ (великий поліном).

Другий поліном складається з N елементів, кожний елемент за модулем 3 і приймає значення 0, 1, -1 (малий поліном). Кількість ненульових елементів фіксована, є параметром алгоритму. Позначимо її T. В деяких алгоритмах кількість 1 та -1 співпадає, для деяких вона може бути різною. У даній роботі розглядається більш загальний випадок, коли фіксується тільки загальна кількість 1 та -1.

Алгоритм 1

Вхід: $a(x)=a_0+a_1x+a_2x^2+\ldots+a_{n-1}x^{N-1}$

Вхід: $b(x)=b_0+b_1x+b_2x^2+\ldots+b_{n-1}x^{N-1}$($b_i \in \{0, 1, -1\}$, кількість ненульових елементів T

Вихід: $c(x)=a(x)*b(x)$

1 $c_0=c_1=c_2=\ldots=c_{2n-1}=0$;

2 Для i:=0 до N–1 виконати:

2.1 Якщо b_i==1, то

2.1.1 Для j:=0 до *N*–1 виконати:

2.1.1.1 $C_{i+j}=C_{i+j}+a_j$.

2.2 Якщо b_i==-1, то

2.2.1 Для j:=0 до *N*–1 виконати:

2.2.1.1 $C_{i+j}=C_{i+j}-a_j$.

Недолік алгоритму. Наявність команд умовного переходу. По кількості непередбачених переходів, які впливають на час виконання операції, можна визначити додаткову інформацію про поліном b.

Для виключення операцій переходу будемо задавати поліном b як структуру, в якій визначимо:

індекси одиничних та мінус одиничних елементів;

кількість одиничних елементів.

Алгоритм 2

Вхід: $a(x)=a_0+a_1x+a_2x^2+\ldots+a_{n-1}x^{N-1}$

Вхід: структура з елементами: *ones* – індекси ненульових елементів, спочатку ідуть індекси одиничних, а потім індекси мінус одиничних елементів, розмір цього масиву фіксований (T). *onescount* – кількість одиничних елементів, тоді кількість елементів, які дорівнюють -1 *minusonescount=T–onescount*, далі Структура

Вихід: c(x)=a(x)*b(x)

1 $c_0=c_1=c_2=\ldots=c_{2N-1}=0$;

2 Для i:=0 до onescount–1 виконати:

2.1 k:=$ones_i$;

2.2 Для j:=0 до *N*–1 виконати:

2.2.1 $c_{k+j}=(c_{k+j}+a_j) \bmod q$

3 Для i:=onescount до *T*–1 виконати:

3.1 k:=$ones_i$;

3.2 Для j:=0 до *N*–1 виконати:

3.2.1 $c_{k+j}=(c_{k+j}-a_j) \bmod q$

Операції додавання та віднімання можна розглядати як однакові операції з погляду обчислювальної складності.

Алгоритм потребує *T* операцій додавання поліномів, а з урахуванням степені поліному *T*N* елементарних операцій зсув на *k* елементів не потребує додаткового часу.

Елементарні операції можуть бути замінені блочними операціями.

Для усіх сучасних NTRU алгоритмів значення $q<2^{14}$, тобто коефіцієнти поліному *a* менше, ніж 2^{14}, тому для коефіцієнту можна виділяти 16 бітів. У разі використання AVX операцій один блок складається з 16 коефіцієнтів.

Позначимо кількість блоків для завдання поліному $m=\left\lceil \frac{N}{16} \right\rceil$.

Позначимо поліном *A*, який відповідає поліному *a* і складається з блоків:

$$A(X)=A_0+A_1x^{16}+\ldots+A_{m-1}x^{16(m-1)}. \quad (9)$$

Поліном *b*(*x*) задаємо у вигляді структури як для Алгоритму 2

Результат – поліном

$$C(X)=C_0+Cx^{16}+\ldots+C_{2m-2}x^{16*(2m-2)} \quad (10)$$

Алгоритм 3

Вхід: $A(X)=A_0+A_1x^{16}+\ldots+A_{m-1}x^{16(m-1)}$

Вхід: Структура

Вихід: C(x)=A(x)*b(x)

1 $C_0=C_1=\ldots=C_{2m-2}=0$;
2 Для i:=0 до onescount–1 виконати:
2.1 k:=$ones_i$;
2.2 Обчислити адресу блоку результату *PC*, це адреса *k* елементу в поліномі C
2.3 Для j:=0 до *m*–1 виконати:
2.3.1 $PC_j=PC_j+A_j \bmod q$
3 Для i:=onescount до *T* виконати:
3.1 k:= $ones_i$;
3.2 Обчислити адресу блоку результату *PC*, це адреса *k* елементу в поліномі C
3.3 Для j:=0 до m–1 виконати:
3.3.1 $PC_j=PC_j–A_j \bmod q$

Недолік алгоритму 3.

Операції додавання та віднімання блоків фактично складаються з операцій: завантаження в регістр, виконання потрібної операції з записом результату в регістр, операція запису блоку з регістру в пам'ять. Адреса PC_j визначається номером ненульового елементу, вона може бути вирівняна на границю блоку, тобто ділитися на 16, або не вирівняною. Для сучасних процесорів звертання до не вирівняної та до вирівняної пам'яті відрізняється несуттєво при читанні даних, але запис по не вирівняній адресі може бути використано для отримання інформації про поліном *b*.

Для виключення звертання до не вирівняної пам'яті в режимі запису модифікуємо алгоритм 3 у вигляді алгоритму 4.

Алгоритм 4

Вхід: $A(X)=A_0+A_1x^{16}+\ldots+A_{m-1}x^{16(m-1)}$

Вхід: Структура

Вихід: C(x)= A(x)*b(x)

1. $C_0=C_1=\ldots=C_{2m-2}=0$;
2. Передобчислення: $P_0=A_0$; $P_1=P_0<<1$; $P_2=P_1<<1$;…$P_{15}=P_{14}<<1$; (Операція << означає операцію зсуву вліво на 1 елемент блоку в сторону старших елементів)
3. Для i:=0 до onescount–1 виконати:
3.1 k:=$ones_i$;
3.2 l=k mod 16
3.3 nb=k/16
3.4 $C_{nb}=C_{nb}+P_k$
3.5 Обчислити початкову адресу для поліному B, з урахуванням передобчислень PB=b–l
3.6 Для j:=1 до *m* виконати
3.6.1 $C_{nb+j}=(C_{nb+j}+PB_j) \bmod q$
4. Для i:=onescount до *T* виконати:
4.1 k:=$ones_i$;
4.2 l=k mod 16
4.3 nb=k/16
4.4 $C_{nb}=C_{nb}–P_k$
4.5 Обчислити початкову адресу для поліному B, з урахуванням передобчислень PB=b–l
4.6 Для j:=1 до *m* виконати
4.6.1 $C_{nb+j}=(C_{nb+j}–PB_j) \bmod q$

В алгоритмі 4 не вирівняна пам'ять використовується тільки для звертання до другого поліному в режимі читання. Звертання до поліному результату і в режимі читання, і в режимі запису виконується для вирівняної пам'яті.

Розглянемо операцію обчислення за модулем *q*.

Для усіх сучасних версій NTRU $q<2^{14}$, тобто при додаванні таких елементів, отримаємо значення, яке не перевищує $2^{15}–1$, тобто цілих довжиною 2 байти. Для приведення за модулем достатньо виконати для кожного коефіцієнту:

При додаванні:

$c = a + b;$ якщо $c>q-1$, то $c=c–q$

і при відніманні:

$c=a–b$; якщо $c<0$, то $c=c+q$.

Визначимо кількість блочних операцій для алгоритму 4.

Для операції передобчислень для значення P0 використовується операція копіювання, яка складається з 2 блочних операцій, для зсуву на 1 елемент використовуються дві операції _mm256_alignr_epi8, _mm256_permute2x128_si256 та операція запису в пам'ять, загальна кількість операцій 2+15*3=47. Усі операції виконуються для вирівняних даних. Кількість операцій не залежить від степені поліному.

Для обчислення значення модуля необхідна операція порівняння, виділення, додавання (віднімання) та запису в пам'ять, тобто 4 блочних операції.

Для обчислення C для одного ненульового значення необхідно завантаження поточного блоку для поліному C (вирівняний блок), завантаження поточного блоку для поліному A (не вирівняний блок), додавання (віднімання) та обчислення модуля для кожного результату. Загальна кількість блокових операцій для коефіцієнтів поліному C – $7mT$

Загальна кількість блокових операцій для множення поліномів дорівнює $7mT+47$.

Для $n=761$, $m=48$, $T=286$ загальна кількість блокових операцій дорівнює 96143 операції.

Паралелізація алгоритму

Для паралельного виконання алгоритму 4 необхідно забезпечити достатнє навантаження ядер (грануляція), рівномірний розподіл навантаження між ядрами (балансування).

Максимальна кількість паралельних потоків визначена експериментально.

В сучасних алгоритмах NTRU не фіксується кількість позитивних та негативних елементів, фіксованою є їх загальна кількість. Для балансування кожний потік виконує операції над позитивними і негативними елементами. Якщо кількість потоків дорівнює ThreadsCount, то кожний потік виконує обробку наступних $\left\lceil \frac{onescount}{TheadsCount} \right\rceil$ та $\left\lceil \frac{minuscount}{TheadsCount} \right\rceil$ ненульових елементів. Останній потік оброблює решту елементів.

В якості вхідних даних для потоку задаються 2 структури.

Структура 1 – загальна для усіх потоків, яка містить поліном A, та передобчислені значення P.

Структура 2 – для кожного з потоків містить адреси початку індексів одиничних та мінус одиничних елементів, та їх кількість.

Кожний потік виконує фактично алгоритм 4 за умови, що йому в якості вхідних даних передаються передобчислені значення (структура 1).

Використання паралельних обчислень без врахування накладних витрат дозволяє отримати обчислювальну складність в разі використання 4 потоків в 24024 блокових операцій.

Для збільшення ефективності критичний код написано на асемблері. Застосування асемблеру дозволило мінімізувати вплив звернення до не вирівняної пам'яті навіть при читанні. При зверненні до не вирівняної пам'яті застосовується додаткова команда звернення до вирівняної і навпаки. Таким чином, кількість звернень і до вирівняної і до не вирівняної пам'яті залишається постійною.

3. Експериментальне дослідження алгоритмів

Мета експериментальної перевірки – визначити експериментально обчислювальну складність запропонованого алгоритму та порівняти її з обчислювальною складністю інших авторів за однакових або близьких параметрів.

Перевірити стійкість до атаки, пов'язаної зі сторонніми каналами запропонованого алгоритму за умови збереження константної кількості ненульових елементів в малому поліномі.

Вихідні дані щодо експериментального дослідження складності множення поліномів для стандарту NTRU та алгоритмів, які запропоновані в якості претендентів на постквантові стандарти, наведено в табл. 1.

Таблиця 1

Вихідні дані щодо дослідження складності множення поліномів

Назва алгоритму	Поле	Параметри
ANSI X9.98 – 2010	$Z/qZ[X](X^N-1)$	N=401..1499, q=2048, t=38..157
NTRU Prime	$Z/qZ[X](X^N-X-1)$	N=439..1021, q – просте, t=18..204

В якості параметрів обрано параметри, для яких автори NTRU Prime навели свою реалізацію (функція rq_mult) і має сенс виконати порівняння двох реалізацій, а саме N=761, q=4591 та t=143 [2] за допомогою одного процесору і середовища. Крім того, вперше наведено результати для параметрів, визначених для криптостійкості K=512, а саме N=1471, t=255, q=12269.

Методика експерименту. При обранні методики експерименту використано рекомендації [15]. Для виміру часу використовуються такти процесору. Для зменшення впливу випадкових факторів, а саме переключення на виконання модулів ОС, завантаження коду для додатку та даних в кеш, і т.д., кожний тест виконується PROBS=100 разів, з цих вимірів обирається мінімальне, тому що збільшення часу можливо тільки в разі додаткового впливу випадкових факторів.

Експериментальне дослідження виконувалось для 10000 ключів [6] з використанням процесору Intel(R) Core (TM) i5-4440 CPU @3.10 GHz.

Створено 3 набори ключів, кожний складається з 10000 ключів. Для визначення впливу не вирівняних даних на час виконання штучно формуються малі поліноми, в яких в якості індексів ненульових елементів обирається максимальна кількість індексів, які кратні 16, що забезпечує мінімізацію звернень до не вирівняної пам'яті в AVX-2 командах. Такий набір ключів позначається GOOD. Другий набір ключових даних формується таким чином, щоб усі індекси ненульових елементів були не кратні 16, тобто кількість звернень до не вирівняної пам'яті була максимальною. Такий набір ключів позначається BAD. В третьому наборі даних індекси ненульових елементів обирались випадково, як це необхідно при генерації ключів. Такий набір ключів позначається RAND.

Для визначення точності вимірів і обчислювальної складності створюється додатковий набір ключів, який позначено EQUAL. Він складається з однакових ключів (перший ключ з набору RAND), саме для цього ключа визначається обчислювальна складність та діапазон, в якому змінюються ці значення для функцій, які аналізуються.

Для кожного набору ключових даних проводяться 10 іспитів.

Спочатку наведемо дані про часову складність методів, а потім – результати дослідження можливості використання часу як додаткової атаки на особистий ключ. Для порівняння з нашими результатами використовується функція rq_mult, яка представлена в алгоритмі NTRU Prime [2]. Будемо називати цю функцію базовою. Саме ця функція використовується в якості базової тому, що вона має найкращі результати щодо обчислювальної складності з відомих в літературі, коли не використовуються спеціальні форми завдання малого поліному (добуток), а також вона доступна в вихідних кодах.

Часові характеристики функцій наведені в табл. 2 (Linux). Для виміру використовується набір ключів EQUAL. Похибка виміру для базової функції менше похибки виміру для нашої функції. Це ускладнює задачу криптоаналітиків використати різницю у вимірі часу як побічного каналу.

Таблиця 2

Часові характеристики функцій множення поліномів (тактів)

K	rq_mult	Точність виміру	Mul	Точність виміру
256	27160- 27860	2.6%	13120-13677	4.24%
512	-	-	30376-31200	2.7%

Таким чином, запропонована функція в середньому забезпечує прискорення в порівнянні з базовою практично в два рази. На жаль, ми не змогли порівняти продуктивність функцій для криптостійкості K=512, тому що реалізація функції rq_mult в режимі оптимізації не передбачає використання інших параметрів. Отримані результати показали, що точність вимірів дещо знижується при використанні нашої функції з 2,5 для базової функції до 4,24 % для нашої.

Результати для 10 серій тестів з наборами GOOD, BAD, RAND представлено в табл. 3. Для кожної серії вимірів задані: мінімальне значення часу (колонка min), номер ключа, для якого це мінімальне значення отримано (nmin), максимальний вимір часу (max).

Таблиця 3

Порівняльний аналіз різних груп ключових даних

Номер серії	Набір ключів	K=256, такти			K=512, такти		
		min	nmin	max	min	nmin	max
1	GOOD	13136	7443	13661	29484	4443	31556
	BAD	13180	412	13727	30280	6174	32138
	RAND	13120	3270	13677	30376	31	31200
2	GOOD	13036	3950	13661	29516	8290	31515
	BAD	13196	3846	13734	30332	8552	32119
	RAND	13160	713	13681	29896	6738	31961
3	GOOD	13148	2740	13659	29472	4443	31503
	BAD	13168	2066	13731	30280	7868	32147
	RAND	13128	2886	13681	29936	1158	32024
4	GOOD	13144	3210	13657	29504	8387	31548
	BAD	13148	8603	13726	30076	7786	32151
	RAND	13084	3283	13681	30188	7740	32005
5	GOOD	13124	4243	1366	29352	5956	31537
	BAD	13216	8174	13731	30316	6407	32127
	RAND	13192	4124	13686	30144	827	31981
6	GOOD	13124	5074	13657	29380	3998	31529
	BAD	13204	5303	13730	30304	3773	32127
	RAND	13148	6107	13683	29972	7061	31986
7	GOOD	13088	6254	13664	29516	4443	31549
	BAD	13132	0	13730	30528	277	32141
	RAND	13128	2322	13676	29872	1158	32022
8	GOOD	13100	4691	13663	29424	550	31526
	BAD	13188	7845	13728	30104	6212	32141
	RAND	13180	3337	13678	30108	2544	31982
9	GOOD	13144	5212	13662	29428	5956	31539
	BAD	13120	208	13725	30400	3803	32132
	RAND	13152	2813	13678	30164	6738	32008
10	GOOD	13104	0	13660	29676	2620	31550
	BAD	13184	1204	13726	30300	3539	32149
	RAND	13128	7531	13679	30216	6753	32009

Як показують результати табл. 3, номери ключів, при яких кількість тактів мінімальна, не співпадають для різних серій (значення 0 відповідають різним групам ключів). Таким чином, використання цього фактору для аналізу ключів неможливе. Мінімальне значення часу може бути як для набору ключів GOOD, так і для інших наборів; таким чином, мінімізація кількості звернень до не вирівняної пам’яті не завжди приводить до зменшення часу. Крім того, для отримання таких ключів необхідно, щоб елементи з індексами 0, 16, 32,…, *N* були

ненульовими. Різниця між значеннями часу для наборів GOOD та BAD не перевищує 2.5% – це мінімальна похибка вимірів. Таким чином, використання цих результатів для атаки сторонніми каналами не ефективне. В той же час алгоритм, запропонований в роботі, дозволяє отримати суттєве підвищення продуктивності функції множення. Результати були підтверджені за допомогою статистичної обробки методом Welch's *t*-test [16].

Реалізацію функцій множення для звичайної форми завдання поліномів та форми f1*f2+f3 (PRODUCT) наведено в [18].

Висновки

1. В асиметричних постквантових криптографічних перетвореннях необхідне суттєве збільшення довжин загальних параметрів, асиметричних пар ключів та електронних підписів тощо. Збільшення довжини параметрів та ключів приводить до збільшення часу виконання операцій; у випадку, що розглядається в статті, – до збільшення складності множення поліномів. При переході з 256 бітної до 512 бітної криптостійкості час виконання функції множення збільшується більше, ніж в два рази (див. табл. 2). Так як операція множення поліномів є складовою частиною генерації ключів, прямого та зворотного перетворень, то вона впливає на складність в цілому. Тому збільшення продуктивності цієї операції є дуже важливим.

2. Запропоновані методи оптимізації дозволили отримати функцію множення поліномів, яка дозволяє збільшити продуктивність практично в два рази по відношенню до найбільш продуктивної функції множення, яка є складовою частиною алгоритму NTRU Prime – учасника конкурсу NIST.

3. Виконано аналіз ефективності атаки, пов'язаної зі сторонніми каналами, а саме часу виконання. За допомогою експериментальних досліджень та їх статистичної обробки показано, що для запропонованого алгоритму ця атака не ефективна.

4. Результати проведених експериментів показали, що для запропонованого алгоритму множення поліномів різниця в складності (часі) для найкращих та найгірших ключів (табл. 3) складає не більше 1 %. В той же час похибка виміру складності множення поліномів для конкретного ключа складає не менше 2 %. Тому використання різниці між складністю множення поліномів для різних ключів для успішної атаки сторонніми каналами є практично не можливим.

5. Отримані результати дозволяють реалізувати захищеність зі значеннями 512 біт класичної та 256 біт квантової криптостійкості.

Список літератури:

1. Lily Chen Stephen Jordan Yi-Kai Liu Dustin Moody Rene Peralta Ray Perlner Daniel Smith-Tone. Report on Post-Quantum Cryptography. [Electronic resource]. Access mode: https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf.

2. Проекти, які прийняті на конкурс. [Electronic resource]. Access mode: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.

3. Daniel J. Bernstein Johannes Buchmann Erik Dahmen. Post-Quantum Cryptography. [Electronic resource]. Access mode: https://www.researchgate.net/profile/Nicolas_Sendrier/publication/226115302_Code-Based_ Cryptography/links/540d62d50cf2df04e7549388/Code-Based-Cryptography.pdf.

4. American National Standard for Financial Services ANSI X9.98 – 2010. Lattice-Based Polynomial Public Key Establishment Algorithm for the Financial Services Industry. Date Approved: 10/15/2010.

5. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime. [Electronic resource]. Access mode: https://ntruprime.cr.yp.to/ntruprime-20160511.pdf.

6. Wei Dai, William Whyte, and Zhenfei Zhang. Optimizing polynomial convolution for NTRUEncrypt. [Electronic resource]. Access mode: https://eprint.iacr.org/2018/229.pdf.

7. Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky and others. CRYSTALS – Dilithium: Digital Signatures from Module Lattices. [Electronic resource]. Access mode: https://eprint.iacr.org/2017/633.pdf.

8. Andreas Hülsing, Joost Rijneveld, John Schanck , and Peter Schwabe. High-speed key encapsulation from NTRU. [Electronic resource]. Access mode: https://cryptojedi.org/papers/ntrukem-20170627.pdf.

9. Toom 3-Way Multiplication. [Electronic resource]. Access mode: https://gmplib.org/manual/Toom-3_002dWay-Multiplication.html.

10. Erdem Alkim, Leo Ducas, Thomas Poppelmann, Peter Schwabe. Postquantum key exchange. A new hope, 2016. [Electronic resource]. Access mode: https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_alkim.pdf.

11. Number-theoretic transform (integer DFT). [Electronic resource]. Access mode: https://www.nayuki.io/page/number-theoretic-transform-integer-dft.

12. Daniel J. Bernstein1, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime: reducing attack surface at low cost. [Electronic resource]. Access mode: https://eprint.iacr.org/2016/461.pdf.

13. Intel 64 and IA-32 Architectures Optimization Reference Manual. [Electronic resource]. Access mode: https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf.

14. Discrete Fourier transform (DFT). https://en.wikipedia.org/wiki/Discrete_Fourier_transform_(general).

15. Oscar Reparaz, Josep Balasch and Ingrid Verbauwhede. Dude, is my code constant time? https://eprint.iacr.org/2016/1123.pdf.

16. Welch's t-test. https://academic.oup.com/biomet/article-abstract/34/1-2/28/210174?redirected From=fulltext.

17. Kachko O., Gorbenko Yu., Yesina M., Akolsina O. Asymmetric Encryption Algorithm Optimization Based on Using NTRU Prime Mathematics // Radiotechnika. 2017. №191. P. 5–10.

18. Kachko O., Gorbenko I., Yesina M., Kandiy S. Polynomials multiplication functions for ordinary and product form of one of the polynomials representation. [Electronic resource]. Access mode: https://github.com/kandiyiit/NTRU-POLYNOMIALS-MULTIPLICATION.

Харківський національний
університет радіоелектроніки;
АТ «Інститут інформаційних технологій»;
Харківський національний
університет імені В. Н. Каразіна

Надійшла до редколегії 09.01.2020