НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

«На правах рукопису»
УДК 004.056.55

«До захисту допущено»
Завідувач кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2025 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Математичні методи криптографічного захисту інформації»

зі спеціальності: 113 Прикладна математика
на тему: **«Модифікація схеми постквантового цифрового підпису «Вершина» з метою підвищення стійкості до атак за побічним каналом»**

Виконав:
студент IV курсу, групи ФІ-13
Мельник Євгеній Ігорович _____

Керівник:
к.ф.-м.н., ст. викладач кафедри ММЗІ
Фесенко Андрій В'ячеславович _____

Рецензент:
к.т.н., доцент
Стьопочкіна Ірина Валеріївна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ — 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти — перший (бакалаврський)
Спеціальність — 113 Прикладна математика,
ОПП «Математичні методи криптографічного захисту інформації»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2025 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Мельник Євгеній Ігорович

1. Тема роботи: «*Модифікація схеми постквантового цифрового підпису «Вершина» з метою підвищення стійкості до атак за побічним каналом*», науковий керівник дипломної роботи: к.ф.-м.н., ст. викладач кафедри ММЗІ Фесенко Андрій В'ячеславович,

затверджені наказом по університету №__ від «__» _____ 2025 р.

2. Термін подання студентом роботи: «__» _____ 2025 р.

3. Об'єкт дослідження: процес перетворення інформації в схемі постквантового цифрового підпису «Вершина».

4. Предмет дослідження: стійкість схеми постквантового цифрового підпису «Вершина» до атак за побічним каналом.

5. Перелік завдань:

1) провести аналіз Dilithium-подібних схем цифрового підпису.

2) здійснити порівняльний аналіз схеми цифрового підпису «Вершина» до схеми підпису Dilithium.

3) розробити та впровадити модифікацію схеми цифрового підпису «Вершина» для підвищення стійкості до атак за побічним каналом.

4) розробити та впровадити модифікацію схеми цифрового підпису «Вершина» з метою зменшення обчислювальної складності процедури підпису.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: Презентація доповіді.

7. Орієнтовний перелік публікацій: планується доповідь на всеукраїнській конференції

8. Дата видачі завдання: 10 вересня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2024 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2024 р.	Виконано
3	Аналіз та порівняння схеми цифрового підпису «Вершина» з схемою підпису Dilithium.	Листопад 2024 р.	Виконано
4	Аналіз методів захисту від атак за побічним каналом у постквантових цифрових підписах (на прикладі Raccoon, qTESLA, HAETAE)	Грудень 2024 р.	Виконано
5	Модифікація схеми цифрового підпису «Вершина» (додавання шуму в секретні параметри, відмова від процедури відсічення)	Січень-лютий 2025 р.	Виконано
6	Оптимізація обчислювальної складності процедури підпису в схеми підпису «Вершина» із використанням техніки передчасного обчислення	Березень-квітень 2025 р.	Виконано
7	Оформлення дипломної роботи	Травень 2025 р.	Виконано

Студент _____ Євгеній МЕЛЬНИК

Керівник _____ Андрій ФЕСЕНКО

РЕФЕРАТ

Кваліфікаційна робота містить: 51 стор., 2 рисунки, 17 джерел.

У дослідженні проаналізовано сучасні схеми постквантового цифрового підпису, зокрема Dilithium-подібні рішення, а також національний стандарт України — схему цифрового підпису «Вершина». Описано особливості побудови та функціонування схеми підпису «Вершина», а також її порівняння з еталонною схемою підпису Dilithium. Проведено огляд і аналіз основних підходів до захисту цифрових підписів від атак за побічним каналом, зокрема розглянуто відомі модифікації сучасних постквантових схем (Raccoon, qTESLA, HAETAE). Запропоновано та обґрунтовано власні модифікації до схеми підпису «Вершина», що передбачають додавання випадкового шуму до секретних параметрів, усунення механізмів помилкових відмов та оптимізацію етапу підпису із використанням стратегії передчасного обчислення. Показано, що ці модифікації підвищують стійкість схеми до атак за побічним каналом і забезпечують гнучку адаптацію під різні рівні криптостійкості без зниження ефективності.

Метою дослідження є побудова модифікації схеми постквантового цифрового підпису «Вершина» з кращою оцінкою стійкості до атак за побічним каналом.

Об'єктом дослідження є процес перетворення інформації в схемі постквантового цифрового підпису «Вершина».

Предметом дослідження є стійкість схеми постквантового цифрового підпису «Вершина» до атак за побічним каналом.

СХЕМА ЦИФРОВОГО ПІДПИСУ «ВЕРШИНА», ПОСТКВАНТОВА КРИПТОГРАФІЯ, АТАКА ЗА ПОБІЧНИМ КАНАЛОМ

ABSTRACT

Qualification thesis consists of: 51 pages, 2 figures, 17 references.

The research analyses modern post-quantum digital signature schemes, including Dilithium-like solutions, as well as the national standard of Ukraine — the digital signature scheme «Vershina». The peculiarities of the design and functioning of the «Vershina» scheme are described, along with its comparison to the reference Dilithium scheme. An overview and analysis of main approaches to protecting digital signatures from side-channel attacks is presented, including a review of well-known modifications of modern post-quantum schemes (Raccoon, qTESLA, HAETAE). The author's own modifications to the «Vershina» scheme are proposed and justified: these include adding random noise to secret parameters, eliminating false rejection mechanisms, and optimizing the signing procedure by using early evaluation strategies. It is shown that these modifications improve the side-channel resistance of the scheme and provide flexible adaptation to different cryptographic security levels without loss of performance.

The aim of the research is to develop a modification of the post-quantum digital signature scheme Vershina with improved resistance to side-channel attacks.

The object of the research is the process of information transformation in the Vershina post-quantum digital signature scheme.

The subject of the research is the resistance of the Vershina post-quantum digital signature scheme to side-channel attacks.

«VERSHINA» DIGITAL SIGNATURE SCHEME, POST-QUANTUM CRYPTOGRAPHY, SIDE-CHANNEL ATTACK

ЗМІСТ

ВСТУП

Актуальність дослідження. Розвиток квантових технологій створює реальну загрозу для класичних криптографічних алгоритмів, що широко застосовуються для захисту інформації. Це зумовлює необхідність впровадження постквантових криптографічних схем, стійких до атак у квантовій моделі обчислень. Особливу роль серед них відіграють алгоритми, що ґрунтуються на складності задач на основі решіток, які сьогодні вважаються перспективними з точки зору захисту від квантових атак.

Національний стандарт України — схема постквантового цифрового підпису «Вершина» — є одним із провідних рішень для впровадження в критичних інформаційних системах. Однак, не менш важливими, ніж математична стійкість, є практичні аспекти захисту від атак за побічним каналом та ефективність реалізації. Саме тому вдосконалення стійкості «Вершини» до таких атак і оптимізація її роботи мають ключове значення для забезпечення надійної інформаційної безпеки в умовах постквантової ери.

Метою дослідження є побудова модифікації схеми постквантового цифрового підпису «Вершина» з кращою оцінкою стійкості до атак за побічним каналом. Для досягнення мети необхідно вирішити такі завдання:

1) провести огляд сучасних постквантових схем підпису та відомих атак на них, зокрема побічних.

2) проаналізувати структуру схеми підпису «Вершина» та визначити потенційні місця для покращення.

3) запропонувати модифікації, спрямовані на підвищення стійкості до атак за побічним каналом.

4) зменшити обчислювальну складність процедури підпису шляхом застосування оптимізації з використанням попередніх обчислень.

Об'єктом дослідження є процес перетворення інформації в схемі постквантового цифрового підпису «Вершина».

Предметом дослідження є стійкість схеми постквантового цифрового підпису «Вершина» до атак за побічним каналом.

Методи дослідження: методи лінійної алгебри, теорія ймовірності, методи моделювання та програмної реалізації криптографічних схем, теорія складності.

Наукова новизна полягає у запропонованих модифікаціях до схеми підпису «Вершина», зокрема:

– застосування додаткового шуму до секретних параметрів для підвищення стійкості до атак за побічним каналом.

– усунення механізмів помилкових відмов (false rejections), які могли спричиняти витоки інформації.

– впровадження оптимізації з використанням попередніх обчислень для зменшення обчислювальної складності підпису.

Практичне значення роботи полягає у підвищенні безпеки реалізацій схеми «Вершина» без погіршення її ефективності, що може бути застосовано у державних інформаційних системах для забезпечення постквантової криптографічної стійкості.

1 ОГЛЯД СХЕМ ПОСТКВАНТОВИХ ЦИФРОВИХ ПІДПИСІВ FALCON ТА DILITHIUM, ЇХ МОДИФІКАЦІЇ ТА АТАКИ

Цей розділ присвячений огляду сучасних схем постквантових цифрових підписів, зокрема схемам Falcon та Dilithium. Розглянуто їхню структуру, відомі модифікації та вразливості, зокрема атаки за побічними каналами, що дозволяє оцінити їхню практичну стійкість.

1.1 Схема постквантового цифрового підпису Falcon

Цифровий підпис Falcon входить до переліку фіналістів конкурсу NIST зі стандартизації постквантової криптографії, який тривав протягом шести років. Схема побудована на основі решіток, а також використовує так звану парадигму «гешування та підпису» (*hash-and-sign*) [10], що є класичним підходом у побудові схем цифрового підпису. Криптографічна стійкість схеми ґрунтується на складності задачі короткого цілочисельного розв'язку (SIS — Short Integer Solution problem) [3], яка є фундаментальною для схем на основі решіток. Якщо спробувати коротко описати, то схема складається з трьох наступних компонентів:

Falcon = фреймворк GPV [6] + решітка NTRU + вибірка за допомогою швидкого перетворення Фур'є (Fast Fourier sampling)

Це поєднання забезпечує високу обчислювальну ефективність, компактність підпису та ключів, що робить Falcon привабливою схемою для впровадження у системах з обмеженими ресурсами.

У контексті переходу від класичних цифрових підписів, таких як

RSA або схем, стійкість яких ґрунтується на складності задачі факторизації чи дискретного логарифмування, до постквантових схем цифрового підпису постає нова проблема — кількість обмінів в протоколі. Зокрема, для більшості постквантових схем характерна відносна простота алгебраїчної структури, однак вони потребують значно більших розмірів відкритих ключів та підписів порівняно з класичними схемами.

Однією з ключових цілей при розробці схеми цифрового підпису Falcon [7] було саме зменшення обсягу даних, які потрібно передавати: як довжини відкритого ключа, так і самого підпису. Саме тому в основі схеми використано решітки, які дозволяють досягти високого рівня компактності без зниження рівня безпеки.

Нехай $n = 2^k$ для деякого $k \in \mathbb{N}$, та зафіксуємо незвідний поліном $\phi(x) = x^n + 1$, а також модуль $q \in \mathbb{N}^*$. Розглянемо кільце

$$R = \mathbb{Z}[x]/(\phi(x)) \tag{1.1}$$

Означення 1.1 (NTRU-рівняння [7])**.** Нехай $f, g, F, G \in R$. Кортеж поліномів (f, g, F, G) задовільняє NTRU-рівнянню, якщо

$$fG - gF \equiv q \mod \phi.$$

Такий кортеж використовується як **секретний ключ**.

Означення 1.2 (Відкритий ключ)**.** Нехай поліном $f \in R$ є оборотним у $R_q := \mathbb{Z}_q[x]/(\phi(x))$. Тоді відкритий ключ визначається як

$$h \leftarrow g \cdot f^{-1} \mod q.$$

Зауваження. Поліноми f і g, як правило, обираються з набору малих коефіцієнтів, наприклад, з $\{-1, 0, 1\}$, для забезпечення низької норми вектора. Водночас h — це поліном з, як правило, великими коефіцієнтами.

Означення 1.3 (NTRU-решітка [7])**.** Визначимо наступні дві матриці над R:

$$\mathbf{B}_{\text{public}} = \begin{bmatrix} 1 & h \\ 0 & q \end{bmatrix}, \quad \mathbf{B}_{\text{secret}} = \begin{bmatrix} f & g \\ F & G \end{bmatrix}.$$

Обидві матриці породжують одну й ту ж решітку, яку називають **NTRU-решіткою**.

Означення 1.4 (NTRU-припущення [7])**.** Нехай $h \in R_q$ є публічним ключем. Тоді задача знайти малі поліноми $f', g' \in R$ такі, що

$$h \equiv g' \cdot (f')^{-1} \mod q$$

є обчислювально складною. Цю складність приймають як основу безпеки NTRU-подібних схем.

У реалізації схеми підпису Falcon використовується каркас GPV, побудований над NTRU-решіткою. Основними параметрами є відкритий та секретний базиси, а також допоміжні значення, необхідні для побудови підпису.

Означення 1.5 (Структура підпису у схемі Falcon [5])**.** Відкритий базис має вигляд матриці

$$\mathbf{A} = \left[\begin{array}{c|c} 1 & h^* \end{array}\right],$$

що еквівалентне знанню полінома h, який виконує роль публічного ключа. Секретний базис, відомий лише підписувачу, задається у вигляді

$$\mathbf{B} = \left[\begin{array}{c|c} f & g \\ \hline F & G \end{array}\right],$$

де поліноми (f, g) та (F, G) пов'язані через NTRU-рівняння. Базиси $\mathbf{A}$ та

$\mathbf{B}$ є ортогональними за модулем q, тобто виконується наступне співвідношення:

$$\mathbf{B} \cdot \mathbf{A}^* \equiv 0 \mod q.$$

Підпис повідомлення m визначається парою (r, s_1, s_2), де r — випадкова сіль, а s_2 — поліном, для якого існує такий s_1, що виконується співвідношення:

$$s_1 + s_2 h = H(r \,\|\, m),$$

де H — геш-функція.

Зауваження. Оскільки поліном s_1 однозначно визначається на основі повідомлення m, солі r та полінома s_2, його не потрібно явно передавати. Таким чином, підпис у схемі FALCON можна компактно представити у вигляді лише пари (r, s_2).

Окрім цього, у схемі `Falcon` фіксується використання циклотомічного полінома $\phi(x) = x^n + 1$, де $n = 2^k$ для деякого натурального k. Такий вибір забезпечує симетричну структуру решітки і дозволяє ефективно застосовувати перетворення Фур'є. Значення модуля q за замовчуванням дорівнює 12289 — воно визначає область визначення всіх поліномів та використовується при обчисленнях у кільці.

Основними елементами в схемі цифрового підпису FALCON є поліноми степеня n з цілими коефіцієнтами, де n — степінь двійки (зазвичай $n = 2^9$ або 2^{10}). Всі обчислення виконуються в кільці $\mathbb{Z}_q[x]/(\phi)$, де $\phi(x) = x^n + 1$ — циклотомічний поліном.

Поліном $f(x)$ розглядається за модулем ϕ як квадратна матриця розміром $n \times n$, рядки якої утворюються як $x^i f(x) \mod \phi$ для всіх i від 0 до $n - 1$. Це дозволяє інтерпретувати операції над поліномами як операції над решітками, породженими відповідними матрицями.

Означення 1.6. Відкритий ключ у схемі FALCON задається

базисом решітки розмірності $2n$ у вигляді матриці

$$\left[\begin{array}{c|c} -h & I_n \\ \hline qI_n & O_n \end{array}\right],$$

де I_n — одинична матриця розміром $n \times n$, O_n — нульова матриця того ж розміру, а h — поліном з коефіцієнтами в $\mathbb{Z}_q$, заданий як матриця у представленні по модулю ϕ.

Означення 1.7. Секретний ключ — це альтернативний базис для подібної решітки, що задається матрицею

$$\left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right],$$

де поліноми $f, g, F, G \in \mathbb{Z}[x]/(\phi)$ задовольняють наступну систему рівнянь:

$$\begin{aligned} h = g/f \quad & \mod \phi \mod q, \\ fG - gF = q \quad & \mod \phi. \end{aligned} \tag{1.2}$$

Створення ключової пари передбачає вибір поліномів f та g з відносно малими коефіцієнтами згідно з певним статистичним розподілом. Після цього, розв'язуючи рівняння NTRU, обчислюються поліноми F та G, що забезпечують правильність секретного базису.

Означення 1.8. Процедура підпису складається з двох етапів. Спочатку обчислюється геш повідомлення m:

$$c = H(r \,\|\, m),$$

де r — випадкова сіль, а H — геш-функція. Далі, використовуючи секретний базис (f, g, F, G), обчислюється пара коротких поліномів (s_1, s_2), таких що

$$s_1 = c - s_2 h \mod \phi \mod q.$$

Підписом вважається лише поліном s_2, оскільки s_1 може бути відновлений під час перевірки.

Означення 1.9. Процедура перевірки підпису полягає в обчисленні гешу $c = H(r \,\|\, m)$ та відновленні $s_1 = c - s_2 h \mod \phi \mod q$. Далі перевіряється, що вектор (s_1, s_2) належить решітці та є досить коротким згідно з наперед визначеним порогом.

Якщо зобразити діаграмою процедуру створення ключової пари то вона буде виглядати приблизно так:

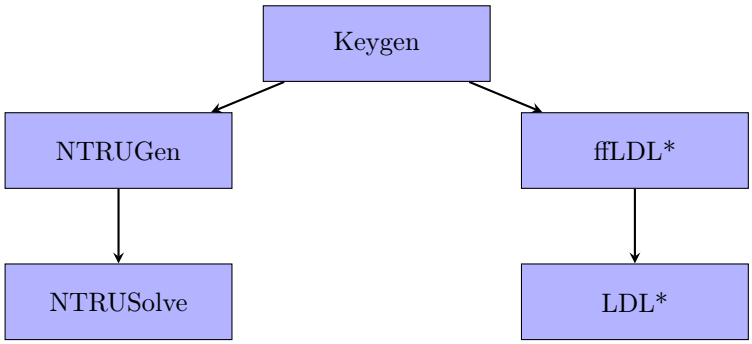

Рисунок 1.1 – Діаграма створення ключової пари

Опишемо псевдокодом як працює функція `Keygen`

Алгоритм 1.1 KEYGEN(ϕ, q)

Вхідні дані: Монічний поліном $\phi \in \mathbb{Z}[x]$ степеня n, модуль q
Результат: Секретний ключ sk, відкритий ключ pk

$f, g, F, G \leftarrow \text{NTRUGEN}(\phi, q)$ ▷ Розв'язання NTRU-рівняння
$\mathbf{B} \leftarrow \begin{bmatrix} g & -f \\ G & -F \end{bmatrix}$ ▷ Матриця з компонентами $g, -f, G, -F$
$\hat{\mathbf{B}} \leftarrow \text{FFT}(\mathbf{B})$ ▷ Обчислення FFT для кожної з 4 компонент
$\mathbf{G} \leftarrow \hat{\mathbf{B}} \times \hat{\mathbf{B}}^{\star}$ ▷ Обчислення добутку матриці на її спряжену транспоновану
$\mathcal{T} \leftarrow \text{FFLDL}^{\star}(\mathbf{G})$ ▷ Обчислення LDL-дерева
for кожен лист $leaf$ у $\mathcal{T}$ **do**
    $leaf.value \leftarrow \sigma / \sqrt{leaf.value}$ ▷ Нормалізація значень у нащадках
end for
$sk \leftarrow (\hat{\mathbf{B}}, \mathcal{T})$ ▷ Створення секретного ключа
$h \leftarrow g f^{-1} \mod q$ ▷ Обчислення публічного ключа h
$pk \leftarrow h$
повернути sk, pk

Детальну реалізацію алгоритмів таких як `NTRUGen`, $ffLDL^*$, LDL^* можна проглянути в оригінальному джерелі [5].

Доцільно також оглянути компоненти, які використовуються в процедурах створення та перевірки підпису.

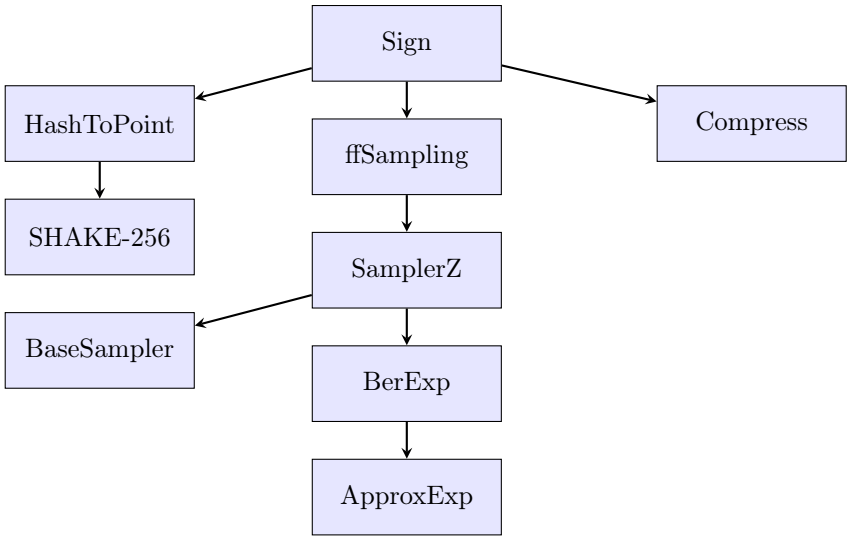

Рисунок 1.2 – Структура алгоритму підпису

Алгоритм 1.2 SIGN$(m, sk, \lfloor\beta^2\rfloor)$

Вхідні дані: Повідомлення m, секретний ключ sk, границя $\lfloor\beta^2\rfloor$
Результат: Підпис sig для повідомлення m

$r \leftarrow \{0,1\}^{320}$ рівномірно розподілена сіль ▷ Згенерувати випадковий вектор довжиною 320
$c \leftarrow$ HASHTOPOINT$(r\|m, q, n)$
$t \leftarrow \left(-\frac{1}{q}\text{FFT}(c) \odot \text{FFT}(F), \frac{1}{q}\text{FFT}(c) \odot \text{FFT}(f)\right)$ ▷ Обчислення t
$t \leftarrow (\text{FFT}(c), \text{FFT}(0)) \cdot \hat{\mathbf{B}}^{-1}$
repeat
  repeat
    $z \leftarrow$ FFSAMPLING$_n(t, \mathcal{T})$ ▷ На цьому етапі z має нормальний розподіл
    $s \leftarrow (t - z)\hat{\mathbf{B}}$
  until $\|s\|^2 \leqslant \lfloor\beta^2\rfloor$ ▷ Оскільки s в FFT-поданні, можна обчислити $\|s\|^2$
  $(s_1, s_2) \leftarrow$ INVFFT(s) ▷ $s_1 + s_2 h = c \mod (\phi, q)$
  $s \leftarrow$ COMPRESS$(s_2, 8 \cdot \text{sbytelen} - 328)$ ▷ Вилучити 1 байт з заголовка та 40 байтів з r
until $s \neq \perp$
повернути $sig = (r, s)$

Детальна реалізація наступних алгоритмів: `HashToPoint`, `ffSampling`, `Compress` та інших наведена в оригінальному джерелі [5].

І в кінці кінців, алгоритм перевірки коректності підпису:

Алгоритм 1.3 VERIFY$(m, sig, pk, \lfloor\beta^2\rfloor)$

Вхідні дані: Повідомлення m, підпис $sig = (r, s)$, відкритий ключ $pk = h \in \mathbb{Z}_q[x]/(\phi)$, границя $\lfloor\beta^2\rfloor$
Результат: Прийняття або відхилення

$c \leftarrow$ HASHTOPOINT$(r\|m, q, n)$
$s_2 \leftarrow$ DECOMPRESS$(s, 8 \cdot \text{sbytelen} - 328)$
if $s_2 = \perp$ **then**
    відхилити ▷ Відхилення некоректного кодування
end if
$s_1 \leftarrow c - s_2 h \mod q$ ▷ s_1 має бути нормалізованим між $\lceil -\frac{q}{2} \rceil$ та $\lceil \frac{q}{2} \rceil$
if $\|(s_1, s_2)\|^2 \leqslant \lfloor\beta^2\rfloor$ **then**
    прийняти
else
    відхилити ▷ Відхилення підписів, які занадто довгі
end if

Детальну реалізацію наступних алгоритмів `HashToPoint` та `Decompress` наведено в оригінальному джерелі [5].

1.2 Схема постквантового цифрового підпису Dilithium

На відміну від FALCON, схема цифрового підпису DILITHIUM була спроєктована з акцентом на простоту реалізації без необхідності роботи у частотній області або використання чисел з плаваючою точкою. Основну увагу під час розробки приділено зручності в апаратній та програмній реалізації, консервативному вибору параметрів, а також мінімізації розміру публічного ключа та підпису при збереженні високого рівня безпеки.

Схема використовує решітки, а також модифікований підхід Фіата–Шаміра з відхиленнями (*Fiat–Shamir with aborts*). Такий підхід дозволяє уникнути передачі випадкових викликів та реалізувати підпис у неінтерактивній формі.

З метою ілюстрації принципів функціонування схеми, нижче наведено псевдокод неефективної (наївної) реалізації підпису, що демонструє її загальну ідею.

Алгоритм 1.4 Створення ключової пари, обчислення підпису і процедура перевірки підпису

Створення ключів (Gen):
$\mathbf{A} \leftarrow R_q^{k \times \ell}$
$(s_1, s_2) \leftarrow S_\eta^\ell \times S_\eta^k$
$\mathbf{t} \leftarrow \mathbf{A}s_1 + s_2$
return $(pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, s_1, s_2))$

Підпис (Sign)(sk, M):
$z \leftarrow \perp$
while $z = \perp$ **do**
    $y \leftarrow S_{\gamma_1 - 1}^\ell$
    $w_1 \leftarrow \textsc{HighBits}(\mathbf{A}y, 2\gamma_2)$
    $c \in B_T \leftarrow \mathrm{H}(M \| w_1)$
    $z \leftarrow y + cs_1$
    if $\|z\|_\infty \geqslant \gamma_1 - \beta$ **or** $\|\textsc{LowBits}(\mathbf{A}y - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geqslant \gamma_2 - \beta$ **then**
        $z \leftarrow \perp$
    end if
end while
return $\sigma = (z, c)$

Верифікація (Verify)$(pk, M, \sigma = (z, c))$:
$w_1' \leftarrow \textsc{HighBits}(\mathbf{A}z - c\mathbf{t}, 2\gamma_2)$
if $\|z\|_\infty < \gamma_1 - \beta$ **and** $c = \mathrm{H}(M \| w_1')$ **then**
    return прийняти
else
    return відхилити
end if

Усі алгебраїчні операції в схемі виконуються над кільцем

$$R_q = \mathbb{Z}_q[X]/(X^n + 1),$$

де $q = 2^{23} - 2^{13} + 1$, $n = 256$. Алгоритм створення ключів створює випадкову матрицю $\mathbf{A} \in R_q^{k \times l}$. Далі випадковим чином вибираються два вектори з малими коефіцієнтами: $\mathbf{s}_1 \in R_q^l$ та $\mathbf{s}_2 \in R_q^k$ та формуються відповідні ключі.

Алгоритм підпису розпочинається з створення випадкового вектора $\mathbf{y} \in R_q^l$, кожен коефіцієнт якого є меншим за параметр γ_1.

Далі обчислюється вектор

$$\mathbf{w}_1 = \mathsf{HighBits}(\mathbf{A} \cdot \mathbf{y}),$$

що містить старші біти коефіцієнтів добутку. На основі повідомлення M та вектора $\mathbf{w}_1$ формується геш-значення $c := H(M \parallel \mathbf{w}_1)$.

Тимчасовий підпис обчислюється як

$$\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1.$$

Для забезпечення незалежності підпису від секретного ключа використовується метод відсічення(rejection sampling): якщо

$$\|\mathbf{z}\|_\infty \geqslant \gamma_1 - \beta \quad \text{або} \quad \mathsf{LowBits}(\mathbf{A}\mathbf{y} - c \cdot \mathbf{s}_2) \notin [-\gamma_2 + \beta, \gamma_2 - \beta],$$

то підпис відкидається, і спроба повторюється.

У разі успіху формується підпис:

$$\sigma := (\mathbf{z}, c).$$

Алгоритм перевірки підпису має на меті впевнитися, що підпис було сформовано коректно, та що він відповідає зафіксованому повідомленню M.

Він починається з обчислення вектора:

$$\mathbf{w}_1' = \mathsf{HighBits}(\mathbf{A}\mathbf{z} - c \cdot \mathbf{t}, 2\gamma_2),$$

де $\mathbf{z}$ та c — елементи отриманого підпису, а $\mathbf{t}$ — частина відкритого ключа.

Далі перевіряються такі умови:

– кожен коефіцієнт вектора $\mathbf{z}$ задовольняє нерівність $|z_i| < \gamma_1 - \beta$;

– значення c збігається з результатом гешування повідомлення та $\mathbf{w}_1'$, тобто:

$$c = H(M, \mathbf{w}_1').$$

Ключовим для коректності перевірки є спостереження, що:

$$\mathsf{HighBits}(\mathbf{A}\mathbf{z} - c \cdot \mathbf{t}, 2\gamma_2) = \mathsf{HighBits}(\mathbf{A}\mathbf{y} - c \cdot \mathbf{s}_2, 2\gamma_2),$$

оскільки:

$$\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1 \quad \text{та} \quad \mathbf{t} = \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2.$$

Підставивши це у вираз, отримаємо:

$$\mathbf{Az} - c \cdot \mathbf{t} = \mathbf{A}(\mathbf{y} + c \cdot \mathbf{s}_1) - c(\mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2) = \mathbf{Ay} - c \cdot \mathbf{s}_2.$$

Таким чином, правильність перевірки зводиться до того, що молодші біти виразу $\mathbf{Ay} - c \cdot \mathbf{s}_2$ не перевищують $\gamma_2 - \beta$, що було перевірено під час підпису (на етапі відбору з відсіченнями*rejection sampling*). Це гарантує, що функція $\mathsf{HighBits}(\cdot, 2\gamma_2)$ дає той самий результат при підписі та верифікації.

Існує два формати реалізації підпису: **детермінований** та **ймовірнісний**. Далі показано саме детермінований підхід. В якості геш функції використовується `SHAKE-256`.

Алгоритм 1.5 Створення ключів

Вхідні дані: Параметри схеми
Результат: Відкритий ключ pk, секретний ключ sk

$\xi \leftarrow \{0,1\}^{256}$
$(\rho, \rho', K) \leftarrow \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} := \mathrm{H}(\xi)$ ▷ Функція H є SHAKE-256
$\mathbf{A} \leftarrow R_q^{k \times \ell} := \textsc{ExpandA}(\rho)$ ▷ Матриця $\mathbf{A}$ створюється та зберігається у формі NTT як $\hat{\mathbf{A}}$
$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k := \textsc{ExpandS}(\rho')$
$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ ▷ Обчислення $\mathbf{t}$
$(t_1, t_0) \leftarrow \textsc{Power2Round}(\mathbf{t}, d)$
$tr \leftarrow \{0,1\}^{256} := \mathrm{H}(\rho \| t_1)$
return $(pk = (\mathbf{A}, t_1), sk = (\mathbf{A}, \tau, \mathbf{t}_1, \mathbf{s}_1, \mathbf{s}_2, t_0))$

Алгоритм 1.6 Верифікація

Вхідні дані: Відкритий ключ pk, повідомлення M, підпис $\sigma = (\tilde{c}, z, h)$
Результат: Прийняти чи відхилити підпис

$\mathbf{A} \in R_q^{k \times \ell} := \textsc{ExpandA}(\rho)$ ▷ Створення та зберігання $\mathbf{A}$ у NTT-поданні як $\hat{\mathbf{A}}$
$\mu \leftarrow \{0,1\}^{512} := \mathrm{H}(\mathrm{H}(\rho \| t_1) \| M)$
$c := \textsc{SampleInBall}(\tilde{c})$
$\mathbf{w}_1' := \textsc{UseHint}_q(h, \mathbf{A}z - ct_1, 2d, 2\gamma_2)$ ▷ Обчислення у NTT-поданні
Перевірити:

$$\|z\|_\infty < \gamma_1 - \beta, \quad [\tilde{c} = \mathrm{H}(\mu \| \mathbf{w}_1')], \quad \#h \leqslant \omega$$

if усі перевірки виконані **then**
    return Прийняти
else
    return Відхилити
end if

Різниця між детермінованим та ймовірнісним підходом лише в 4

Алгоритм 1.7 Підпис

Вхідні дані: Секретний ключ sk, повідомлення M
Результат: Підпис $\sigma = (\tilde{c}, z, h)$

$\mathbf{A} \in R_q^{k \times \ell} := \text{ExpandA}(\rho)$ ▷ Створення та зберігання $\mathbf{A}$ у NTT-поданні як $\hat{\mathbf{A}}$
$\mu \leftarrow \{0,1\}^{512} := \text{H}(\tau \| M)$
$\kappa := 0, (z, h) :=\perp$
$\rho' \leftarrow \text{H}(K \| \mu)$ або $\rho' \leftarrow \{0,1\}^{512}$ для ймовірнісного підходу
while $(z, h) =\perp$ **do** ▷ Попереднє обчислення $\hat{s}_1$, $\hat{s}_2$, $\hat{t}_0$
  $y \leftarrow S^{\ell}_{\gamma_1 - 1} := \text{ExpandMask}(\rho', \kappa)$
  $\mathbf{w} := \mathbf{A}y$
  $\mathbf{w}_1 := \text{HighBits}(\mathbf{w}, 2\gamma_2)$
  $\tilde{c} \leftarrow \{0,1\}^{256} := \text{H}(\mu \| \mathbf{w}_1)$
  $c \leftarrow B_\tau := \text{SampleInBall}(\tilde{c})$ ▷ Зберегти c у NTT-поданні як $\hat{c}$
  $z := y + c\mathbf{s}_1$
  $r_0 := \text{LowBits}(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$
  if $\|z\|_\infty \geqslant \gamma_1 - \beta$ **or** $\|r_0\|_\infty \geqslant \gamma_2 - \beta$ **then**
    $(z, h) :=\perp$
  else
    $h := \text{MakeHint}_q(-ct_0, \mathbf{w} - c\mathbf{s}_2 + ct_0, 2\gamma_2)$
    if $\|\text{ct}_0\|_\infty \geqslant \gamma_2$ **or** кількість одиниць у h перевищує ω **then**
      $(z, h) :=\perp$
    end if
  end if
  $\kappa := \kappa + \ell$
end while
return $\sigma = (\tilde{c}, z, h)$

рядку алгоритму підпису, тобто у виборі ρ'

1.3 Огляд існуючих модифікацій схем Falcon та Dilithium та атаки на них

В даному підрозділі будуть наведені наявні модифікації цифрових підписів Falcon та Dilithium.

Схема `MITAKA` є варіантом цифрового підпису, побудованого на основі NTRU-решіток, та походить від схеми `Falcon`. Основною метою `MITAKA` є спрощення алгоритму підписування, підвищення можливості паралельного виконання обчислень і покращення стійкості до атак за побічними каналами при збереженні ефективності, порівнянної зі схемою `Falcon`. Крім того, `MITAKA` забезпечує вищу гнучкість у виборі параметрів, що робить її більш адаптованою до різноманітних сценаріїв застосування.

Основною відмінністю Mitaka є використання іншого алгоритму семплінгу. Тобто замість ffSampling тепер використовуються інші підходи.

Алгоритм 1.8 Алгоритм обчислення підпису

Вхідні дані: Повідомлення m, секретний ключ sk, межа γ
Результат: Підпис sig для m

do
  $r \leftarrow \{0,1\}^k$
  $c \leftarrow H(r\|m)$
  $z \leftarrow \text{SAMPLER}(sk, (0, c))$ ▷ Використання Sampler замість ffSampling
  $s \leftarrow (s_1, s_2) = (0, c) - z$
while $\|s\|^2 > \gamma^2$
return $sig = (r, s_1)$

Детальна реалізація модифікованих алгоритмів семплінгу описана в специфікації Mitaka [4].

Якщо говорити про Dilithium, то в оригінальній схемі цифрового підпису Dilithium для гешування на етапах створення ключів, підписування та перевірки використовується криптографічна геш-функція SHAKE-256, яка належить до родини алгоритмів SHA-3. Незважаючи на високий рівень криптографічної стійкості та гнучкість, SHAKE-256 наразі не має такої широкої підтримки в апаратному забезпеченні, як інші криптографічні примітиви, зокрема AES.

У зв'язку з цим була запропонована модифікація Dilithium-AES, яка замінює використання SHAKE-256 на широко підтримуваний у сучасних процесорах алгоритм симетричного шифрування AES (у режимі роботи, що імітує геш-функцію). Такий підхід дозволяє суттєво скоротити час виконання основних криптографічних операцій у середовищах із апаратною підтримкою AES, зокрема в процесорах з інструкціями AES-NI. Крім того, це спрощує апаратну реалізацію алгоритму на вбудованих системах, де підтримка SHAKE може бути обмеженою або відсутньою.

Таким чином, Dilithium-AES є оптимізованим варіантом базової схеми, орієнтованим на практичне використання в пристроях з обмеженими ресурсами та в умовах, де критичною є апаратна

ефективність.

Цифрові підписи `Falcon` та `Dilithium` реалізують принципово різні підходи до побудови схем: `Falcon` ґрунтується на фреймворку `GPV` [6], тоді як `Dilithium` використовує евристичну трансформацію Фіата–Шаміра.

Фреймворк `GPV` має строгі докази стійкості як у класичній, так і в квантовій моделях обчислень, оскільки його безпека ґрунтується на складності задачі короткого цілочисельного розв'язку (`SIS`). Проте, попри теоретичну стійкість, реалізації схем на основі `GPV` можуть бути вразливими до атак за побічними каналами, які дозволяють зібрати додаткову інформацію про внутрішні змінні алгоритму [11].

Далі наведено ідею так званої атаки на відновлення секретного ключа у фреймворку `GPV`.

У загальному вигляді підпис у схемі `GPV` визначається співвідношенням [6]:

$$s = (r - \bar{z})\tilde{B},$$

де:

– s — вектор підпису,

– $r = (r_1, r_2, \ldots, r_n)$ — випадковий вектор, згенерований під час процедури підписування,

– $\bar{z} = (\bar{z}_1, \bar{z}_2, \ldots, \bar{z}_n)$ — наближене значення r, отримане через побічні канали,

– $\tilde{B}$ — ортогоналізований секретний базис решітки.

Метою атаки є наближене відновлення секретного базису $\tilde{B}$ (а отже, і самого секретного ключа), використовуючи вектори підписів s_i та спостереження $\bar{z}_i$. Для цього застосовується метод найменших квадратів, що дозволяє побудувати оцінку $\hat{B}$, яка мінімізує відхилення між спостереженнями та моделлю:

$$\min_{\hat{B}} \sum_{i=1}^{N} \left\| s_i - (r_i - \bar{z}_i)\hat{B} \right\|^2,$$

де N — кількість доступних підписів, а $\hat{B}$ — оцінка ортогонального базису.

Аналітичний розв'язок цієї задачі має вигляд:

$$\hat{B} = (W^\top W)^{-1} W^\top S,$$

де:

– W — матриця, рядки якої складаються з векторів $(r_i - \bar{z}_i)$,

– S — матриця, що містить відповідні вектори підпису s_i.

Таким чином, при наявності достатньої кількості підписів та точних побічних спостережень $\bar{z}_i$, зловмисник може з високою ймовірністю відновити структуру секретного базису, що становить серйозну загрозу для реалізацій схем на основі `GPV` без належного захисту.

Висновки до розділу 1

В даному розділі проведено глибокий огляд сучасних постквантових схем цифрового підпису Falcon і Dilithium. Обидві схеми ґрунтуються на решітках, але використовують принципово різні криптографічні підходи: Falcon — GPV-фреймворк з теоретично доведеною безпекою, а Dilithium — евристичну трансформацію Фіата–Шаміра з відхиленнями.

Розглянуто внутрішню структуру схем, алгоритми генерації ключів, підпису та перевірки, а також специфіку використання математичних примітивів — зокрема, циклотомічних кілець і алгоритмів семплінгу.

Особливу увагу приділено існуючим модифікаціям Falcon та Dilithium, зокрема схемам Mitaka та Dilithium-AES, які демонструють спроби підвищити практичну ефективність та стійкість до атак за побічними каналами. Такі модифікації свідчать про активне дослідження напрямку забезпечення практичної стійкості, а не лише теоретичної.

Також розглянуто типові атаки на реалізації, включаючи приклади витоку інформації через побічні спостереження та методи наближеного відновлення секретного ключа на основі GPV-підписів.

2 МОДИФІКАЦІЇ СХЕМИ ЦИФРОВОГО ПІДПИСУ «ВЕРШИНА»

У сучасних криптографічних системах, особливо тих, що реалізуються на вбудованих пристроях, атаки за побічним каналом (Side-Channel Attacks, SCA) становлять серйозну загрозу. Вони дозволяють зловмисникам отримувати конфіденційну інформацію, аналізуючи фізичні характеристики пристроїв, такі як споживання енергії, електромагнітне випромінювання або час виконання операцій. Особливо вразливими є постквантові схеми цифрового підпису, які, незважаючи на математичну стійкість, можуть бути скомпрометовані через недоліки в реалізації.

2.1 Порівняльний аналіз схеми цифрового підпису «Вершина» відповідно до схеми підпису Dilithium

Схема підпису «Вершина» є Dilithium-подібною схемою цифрового підпису, побудованою на алгебраїчних решітках, а також з використанням парадигми Фіат-Шаміра відхиленнями, і розроблена для забезпечення високої ефективності та стійкості до квантових атак, однак має власну специфіку у виборі параметрів, створення ключів та процедурі підпису.

Як і в схемі підпису Dilithium, всі операції відбуваються в кільці $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$, де q — велике просте число, а n — степінь полінома $X^n + 1$.

Параметр λ визначає рівень класичної криптостійкості. У схемі підпису «Вершина» передбачено чотири рівні: $\lambda = 128, 256, 384, 512$. На відміну від схеми підпису Dilithium, яка не підтримує рівень $\lambda = 512$, у схемі підпису «Вершина» реалізовано підвищений рівень безпеки.

Степінь полінома n задає кількість коефіцієнтів у кільці

$R = \mathbb{Z}_q[X]/(X^n + 1)$. Для $\lambda = 128, 256$ використовується $n = 256$, тоді як для $\lambda = 384, 512$ — $n = 512$.

Модуль q є простим числом, що визначає область визначення коефіцієнтів у кільці R_q. Для всіх рівнів безпеки використовується модуль $q = 2^{23} - 2^{13} + 1$, причому $q \equiv 1 \pmod{2n}$.

Параметр *mode* визначає режим застосування, який задає конкретні розміри матриці A та векторів s_1, s_2. Визначено чотири режими:

– режим 0: $\lambda = 128$, $k = 4$, $l = 4$;

– режим 1: $\lambda = 256$, $k = 6$, $l = 5$;

– режим 2: $\lambda = 384$, $k = 7$, $l = 5$;

– режим 3: $\lambda = 512$, $k = 9$, $l = 8$.

Параметр $d = 13$ визначає кількість бітів, які відкидаються при поділі коефіцієнтів полінома t на молодшу та старшу частини.

Параметр η визначає діапазон значень для коефіцієнтів поліномів s_1 та s_2. Його значення залежать від режиму: $\eta = 2$ для режимів 0 і 3, $\eta = 4$ для режиму 1, $\eta = 5$ для режиму 2.

Параметр γ_1 задає межу для коефіцієнтів полінома y. Для режиму 0 використовується $\gamma_1 = 2^{17}$, для інших — 2^{19}.

Параметр γ_2 використовується при перевірці норми перетворень і залежить від режиму. Для режиму 0 обчислюється як $(q - 1)/88$, для інших — як $(q - 1)/32$.

Параметр ω визначає максимальну кількість одиниць у матриці h: $\omega = 80, 55, 150, 230$ відповідно до режиму.

Параметр w_c задає кількість ненульових коефіцієнтів у виклику c: $w_c = 39, 49, 77, 118$ залежно від режиму.

Параметр β визначає межу ℓ_∞-норми вектора z під час перевірки підпису: $\beta = 78, 196, 62, 74$.

Крім цього, визначено параметри, пов'язані з довжиною ключів і підписів:

– `SEED_OCTETS`: довжина випадкового рядка, дорівнює 32 байти для режимів 0 і 1, або `OCTETS`(λ) для режимів 2 і 3;

– $\texttt{HASH\_OCTETS} = \texttt{OCTETS}(2\lambda)$;

– $\texttt{PK\_OCTETS} = \texttt{SEED\_OCTETS} + \texttt{OCTETS}(k \cdot n \cdot (\log_2 q - d))$;

–

$\texttt{SK\_OCTETS} = 2{\cdot}\texttt{SEED\_OCTETS}{+}\texttt{HASH\_OCTETS}{+}\texttt{OCTETS}(n{\cdot}((l{+}k){\cdot}\log_2(2\eta{+}1){+}k{\cdot}d))$;

– $\texttt{DS\_OCTETS} = \texttt{SEED\_OCTETS} + \texttt{OCTETS}(\log_2(2\gamma_1) \cdot n \cdot l) + \omega + k \cdot n/256$.

Таким чином, параметри схеми підпису «Вершина» забезпечують гнучкість у налаштуванні рівня безпеки, що дозволяє адаптувати реалізацію до конкретних вимог користувача.

Створення ключової пари виконується за допомогою функції `ds_gen_key`, яка створює відкритий ключ pk та секретний ключ sk. Алгоритм використовує операції в кільці поліномів $\mathbb{Z}_q[X]/(X^n + 1)$ із застосуванням теоретичного перетворення чисел (Number-Theoretic Transformation, NTT).

Алгоритм 2.1 Функція `ds_gen_key` для створення ключової пари у схемі підпису «Вершина»[17]

Input: випадкові бітові рядки ρ, ρ_1, ключ key
Output: відкритий ключ pk, секретний ключ sk
$ctx \leftarrow$ `AllocContext()` ▷ Створення службового контексту
$A \leftarrow$ `ds_expand_matr`(ρ, ctx) ▷ Розгортання матриці $A \in \mathbb{R}_q^{k \times l}$
$s_1, s_2 \leftarrow$ `ds_expand_s`(ρ_1, ctx) ▷ Обчислення векторів $s_1 \in \mathbb{R}_q^l$, $s_2 \in \mathbb{R}_q^k$
`FreeContext`(ctx) ▷ Звільнення контексту
$s_1 \leftarrow$ `vec_ntt`(s_1, l) ▷ теоретичне пряме перетворення чисел
$t \leftarrow$ `matr_vec_mul`(A, s_1) ▷ Множення $A \cdot s_1$
$t \leftarrow$ `vec_intt`(t, k) ▷ теоретичне зворотнє перетворення чисел
$t \leftarrow$ `vec_add`(t, s_2, k) ▷ Додавання s_2 до t
for $i \in [k \cdot n]$ **do**
    $t_0[i], t_1[i] \leftarrow$ `ds_gen_hl`$(t[i], 2^d)$ ▷ Розбиття коефіцієнтів
end for
$pk, pk\_len \leftarrow$ `ds_pack_pk`(ρ, t_1) ▷ Упаковка відкритого ключа
$tr \leftarrow$ `Hash`(pk, pk_len) ▷ Обчислення гешу відкритого ключа
$sk \leftarrow$ `ds_pack_sk`$(\rho, key, tr, s_1, s_2, t_0)$ ▷ Упаковка секретного ключа
return (pk, sk)

Отже, якщо коротко описати, то на вхід функції подаються три випадкові бітові рядки. Рядок $\rho \in \{0,1\}^{\texttt{SEED\_OCTETS}}$ використовується для створення матриці A. Рядок $\rho_1 \in \{0,1\}^{\texttt{SEED\_OCTETS}}$ слугує для створення векторів s_1 та s_2, які є частиною секретного ключа. Третій параметр `key` також є випадковим рядком довжини `SEED_OCTETS` і застосовується для

подальшого маскування внутрішніх значень у процесі створення секретного ключа.

Виходом функції є відкритий та секретний ключі. Відкритий ключ pk має довжину `PK_OCTETS` і складається з пари (ρ, t_1), де t_1 є частиною декомпозиції вектора $t = A \cdot s_1 + s_2$. Секретний ключ sk має довжину `SK_OCTETS` і містить набір параметрів $(\rho, \texttt{key}, tr, s_1, s_2, t_0)$, де tr — геш відкритого ключа, а t_0 — доповнення до t_1 згідно з внутрішньою структурою алгоритму.

Зауваження. У реалізації алгоритму створення ключової пари можливі варіанти компромісу між ефективністю та обсягом пам'яті:

– Замість збереження рядка байтів `sk` можна зберігати повні вектори s_1, s_2, що прискорює обчислення підпису, проте потребує більше пам'яті.

– Замість збереження s_1, s_2 у складі ключа можна зберігати лише початковий рядок ρ_1, з якого ці вектори відновлюються. Це зменшує розмір секретного ключа, але збільшує час підпису.

Алгоритм створення електронного підпису в схемі підпису «Вершина» допускає два варіанти реалізації: детермінований, у якому підпис є однаковим для однакових повідомлень, та ймовірнісний, у якому для одного й того самого повідомлення підпис щоразу формується знову і є різним. Така гнучкість дозволяє адаптувати схему до різних сценаріїв застосування та забезпечити стійкість до атак повторного підпису.

Параметри алгоритму:

– n — степінь полінома $X^n + 1$, який визначає кількість коефіцієнтів у векторах.

– k, l — розміри векторів s_2 та s_1 відповідно; також задають розміри матриці A.

– η — амплітуда коефіцієнтів поліномів s_1, s_2.

– d — кількість бітів, що використовуються для відсікання молодшої частини коефіцієнтів.

– γ_1 — межа для коефіцієнтів вектора y під час маскування.

– γ_2 — параметр для перевірки обмежень векторів при

трансформаціях.

– β — межа для ℓ_∞-норми вектора z.

– w_c — кількість ненульових коефіцієнтів у векторі виклику c.

– `SEED_OCTETS` — довжина випадкового значення у байтах.

– `HASH_OCTETS` — довжина гешу у байтах.

– `BLOCK_SIZE` — розмір блоку для генератора псевдовипадкових послідовностей.

Вхідні дані:

– m — повідомлення для підпису у вигляді байтового рядка.

– m_{len} — довжина повідомлення m у байтах.

– sk — секретний ключ у форматі рядка байтів довжини `SK_OCTETS`.

– `variant` $\in \{0, 1\}$ — тип підпису. При значенні 0 підпис є детермінованим (однаковим для одного і того ж повідомлення), при 1 — випадковим.

– `ctx` — контекст генератора псевдовипадкових бітів (використовується лише якщо `variant` $\neq 0$).

Вихідні дані:

– ds — сформований електронний підпис у вигляді байтового рядка довжини `DS_OCTETS`.

У схемі цифрового підпису «Вершина» алгоритм підпису ґрунтується на парадигмі Фіат–Шаміра з відхиленнями. Для кожного повідомлення створюється випадковий вектор маскування, на основі якого формується виклик, відповідь та обчислюється допоміжна перевірка. Якщо під час перевірки значення виходять за допустимі межі, підпис відкидається, і процедура повторюється. Останній крок, пов'язаний із відсіченням (rejection sampling), забезпечує правильність, однак є потенційно вразливим до атак за побічним каналом, оскільки витікає інформація про внутрішні обчислення через кількість повторів.

Алгоритм 2.2 Алгоритм створення ЕП в схемі цифрового підпису «**Вершина**»

1: Розпаковка секретного ключа:
$\{\rho, \texttt{key}, tr, s_1, s_2, t_0\} \leftarrow \texttt{ds\_unpack\_sk}(sk)$
2: Обчислення теоретичного перетворення чисел для s_1, s_2, t_0:
$s_1 \leftarrow \texttt{vec\_ntt}(s_1, l)$
$s_2 \leftarrow \texttt{vec\_ntt}(s_2, k)$
$t_0 \leftarrow \texttt{vec\_ntt}(t_0, k)$
3: Формування буфера μ:
$\texttt{buffer} \leftarrow tr \parallel m$
$\mu \leftarrow \texttt{Hash}(\texttt{buffer}, \texttt{HASH\_OCTETS} + m\_len)$
4: Відновлення матриці A за допомогою ρ:
$A \leftarrow \texttt{ds\_expand\_matr}(\rho)$
5: **if** $\texttt{variant} \neq 0$ **then**:
6: $\texttt{rand\_comp} \leftarrow \texttt{ds\_rand\_octets}(ctx, \texttt{SEED\_OCTETS})$
$\texttt{len} \leftarrow \texttt{SEED\_OCTETS}$
7: **else**:
8: $\texttt{len} \leftarrow 0 \quad \texttt{rand\_comp} \leftarrow \perp$
9: Формування буфера та гешу ro_1:
$\texttt{buffer} \leftarrow \texttt{rand\_comp} \parallel \texttt{key} \parallel \mu$
$\texttt{len} \leftarrow \texttt{len} + \texttt{SEED\_OCTETS} + \texttt{HASH\_OCTETS}$
$ro_1 \leftarrow \texttt{Hash}(\texttt{buffer}, \texttt{len})$
10: Ініціалізація ГПВЧ:
$ctx\_y \leftarrow \texttt{PsevdoRandomInit}(ctx\_y, ro_1, \texttt{HASH\_OCTETS})$
11: **repeat**
12: Створення вектора y та теоретичне перетворення чисел:
$ctx\_y, y \leftarrow \texttt{ds\_gen\_y}(ctx\_y)$
$ntt\_y \leftarrow \texttt{vec\_ntt}(y, l)$
13: Обчислення $w := A \cdot y$, а також розбиття w на w_1, w_0:
$ntt\_w \leftarrow \texttt{matr\_vec\_mul}(A, ntt\_y)$
$w \leftarrow \texttt{vec\_ntt}(ntt\_w)$
$w_1[i], w_0[i] \leftarrow \texttt{ds\_gen\_hl}(w[i], 2\gamma_2)$
14: Обчислення c: $seed \leftarrow \texttt{ds\_prepare\_gen\_c}(w_1, \mu)$
$c \leftarrow \texttt{ds\_gen\_c}(seed)$
15: Обчислення $z := y + c \cdot s_1$, перевірка норми: $c \leftarrow \texttt{ntt}(c)$
$z[i] \leftarrow \texttt{pointwise}(c, s_1[i])$
$z \leftarrow \texttt{vec\_ntt}(z, l)$
if $\texttt{ds\_check\_vect}(z, \gamma_1 - \beta, l) = \texttt{ERROR}$ **then** повертаємось на крок 12
16: Обчислення $r := w - c \cdot s_2$, та r_1, r_0: $r[i] \leftarrow \texttt{pointwise}(c, s_2[i])$
$r \leftarrow \texttt{vec\_ntt}(r, k)$
$r \leftarrow \texttt{vec\_sub}(w, r, k)$
$r_1[i], r_0[i] \leftarrow \texttt{ds\_gen\_hl}(r[i], 2\gamma_2)$
17: **if** $\texttt{ds\_check\_vect}(r_0, \gamma_2 - \beta, k) = \texttt{ERROR}$ **then** повертаємось на крок 12
18: Обчислення $ct_0 := c \cdot t_0$, перевірка норми: $ct_0[i] \leftarrow \texttt{pointwise}(c, t_0[i])$
$ct_0 \leftarrow \texttt{vec\_ntt}(ct_0, k)$
if $\texttt{ds\_check\_vect}(ct_0, \gamma_2, k) = \texttt{ERROR}$ **then** повертаємось на крок 12
19: Обчислення $temp := r + ct_0$, перетворення на $temp_1, temp_0$: $temp \leftarrow \texttt{vec\_add}(r, ct_0, k)$
$temp_1[i], temp_0[i] \leftarrow \texttt{ds\_gen\_hl}(temp[i], 2\gamma_2)$
20: Обчислення h, підрахунок кількості ненульових компонент: $h, cnt \leftarrow \texttt{ds\_gen\_h}(r_1, temp_1)$
if $cnt > \omega$ **then** повертаємось на крок 12
21: **until** допустима підписна пара (z, h)
22: Упаковка підпису: $ds \leftarrow \texttt{ds\_pack\_ds}(seed, z, h)$
23: **return** ds

Побудова електронного підпису у схемі підпису «Вершина» ґрунтується на ітераційному процесі, який забезпечує коректність лише за умови проходження всіх перевірок — зокрема перевірок норм та вагових обмежень. Це реалізується шляхом багаторазового повторення циклу обчислення підпису, поки не буде отримано припустимий результат. Безпека алгоритму при цьому забезпечується за рахунок використання геш-функції у моделі випадкового оракула, яка моделює запити з боку гіпотетичного верифікатора.

Схема є ймовірнісною: кожне нове підписування одного й того самого повідомлення використовує нові випадкові дані, що ускладнює повторне використання або підробку підпису.

Водночас, останній етап алгоритму — перевірка норми та ваги з можливим відхиленням (rejection sampling) — є вразливим з точки зору атак за побічним каналом. Кількість ітерацій до отримання припустимого підпису може залежати від секретного ключа, що відкриває можливості для атак за часом виконання, якщо реалізація не забезпечує постійну тривалість виконання незалежно від даних.

З метою підвищення ефективності в алгоритмі активно використовується спеціальний формат завдання поліномів (NTT - Number-Theoretic Transform), що пришвидшує множення поліномів у кільці R_q.

Варто зазначити ще раз, що схема підтримує чотири режими використання (від 0 до 3), що дозволяє обирати бажаний баланс між рівнем безпеки та ефективністю. На відміну від Dilithium, схема підпису «Вершина» реалізує також режим із параметром стійкості $\lambda = 512$, що забезпечує вищий рівень захисту в порівнянні з більшістю стандартних реалізацій.

Алгоритм перевірки електронного підпису у схемі підпису «Вершина» використовує наступні параметри:

– n — степінь полінома, що визначає розмірність простору;

– q — модуль, за яким виконуються всі обчислення в кільці $\mathbb{R}_q$;

– k — розмірність векторів, пов'язаних із матрицею A;

– l — розмірність векторів, пов'язаних із вектором s_1;

– η — амплітуда коефіцієнтів секретних векторів;

– d — кількість бітів для зберігання менш значущих частин векторів;

– γ_1, γ_2 — параметри, що задають межі норм для перевірки;

– w_c — кількість ненульових коефіцієнтів у векторі виклику c;

– β — межа для ℓ_∞-норми вектора z;

– `SEED_OCTETS`, `HASH_OCTETS` — розміри випадкових компонентів та гешу відповідно;

– `BLOCK_SIZE` — розмір блоку, що використовується у внутрішньому перетворенні.

Вхідні дані алгоритму перевірки:

– pk — відкритий ключ, що містить матрицю A та вектор t_1;

– m — повідомлення, що було підписане;

– m_{len} — довжина повідомлення;

– ds — підпис у вигляді рядку довжиною `DS_OCTETS`.

Вихідні дані:

– Алгоритм повертає логічне значення: `TRUE`, якщо підпис є коректним відносно повідомлення m та відкритого ключа pk; `FALSE` — у протилежному випадку.

Якщо коротко описати ідею алгоритму, то він перевіряє, чи є електронний підпис дійсним для повідомлення m за відкритим ключем pk. Для цього він відновлює матрицю A та поліном c, використовуючи значення ρ та $seed$, що входять до складу підпису. Вектор w_1 реконструюється з результату обчислення Az та матриці h, яка була частиною підпису. Цей процес дозволяє повторно отримати значення $seed$, яке порівнюється з оригінальним значенням, що міститься в підписі. У випадку збігу перевірка вважається успішною.

Алгоритм 2.3 Алгоритм перевірки електронного підпису у схемі підпису «Вершина» [17]

Розпакування відкритого ключа:
  $\rho, t_1 \leftarrow \texttt{ds\_unpack\_pk}(pk)$
Розпакування підпису:
  $\texttt{seed}, \mathbf{z}, \mathbf{h} \leftarrow \texttt{ds\_unpack\_ds}(ds)$
Обчислення гешу відкритого ключа:
  $\texttt{hash\_pk} \leftarrow \texttt{Hash}(pk, \texttt{PK\_OCTETS})$
Формування буфера для обчислення μ:
  $\texttt{buffer} \leftarrow \texttt{hash\_pk} \| m$
  $\mu \leftarrow \texttt{Hash}(\texttt{buffer}, \texttt{HASH\_OCTETS} + m\_len)$
Відновлення матриці A та полінома c:
  $A \leftarrow \texttt{ds\_expand\_matr}(\rho)$
  $c \leftarrow \texttt{ds\_gen\_c}(\texttt{seed})$
Обчислення векторів:
  $\mathbf{z} \leftarrow \texttt{vec\_ntt}(\mathbf{z}, l)$
  $c \leftarrow \texttt{ntt}(c)$
  $t_{1d} \leftarrow \texttt{vec\_cmul}(t_1, 2^d, k)$
  $t_{1d} \leftarrow \texttt{vec\_ntt}(t_{1d}, k)$
for $i = 0$ **to** $k - 1$ **do**
  $Az[i] \leftarrow \texttt{vec\_smul}(A[i], \mathbf{z}, l)$
  $t_1[i] \leftarrow \texttt{pointwize}(t_1[i], c)$
  $Az[i] \leftarrow \texttt{pol\_sub}(Az[i], t_1[i])$
  $Az[i] \leftarrow \texttt{ntt}(Az[i])$
end for
Обчислення $\mathbf{w_1}$ і $\mathbf{w_0}$ та корекція:
$cnt \leftarrow q/(2\gamma_2)$
for $i = 0$ **to** $k - 1$ **do**
  $w_1[i], w_0 \leftarrow \texttt{ds\_gen\_hl}(Az[i], 2\gamma_2)$
  for $j = 0$ **to** $n - 1$ **do**
    if $w_0[j] > q/2$ **then**
      $w_0[j] \leftarrow w_0[j] - q$
    end if
  end for
  for $j = 0$ **to** $n - 1$ **do**
    if $h[i][j] \neq 0$ **then**
      if $w_0[j] > 0$ **then**
        $w_1[i][j] \leftarrow w_1[i][j] + 1$
      else
        $w_1[i][j] \leftarrow w_1[i][j] - 1$
      end if
      $w_1[i][j] \leftarrow w_1[i][j] \bmod cnt$
      if $w_1[i][j] < 0$ **then**
        $w_1[i][j] \leftarrow w_1[i][j] + cnt$
      end if
    end if
  end for
end for
Відновлення *seed* **та перевірка:**
$\texttt{calc\_seed} \leftarrow \texttt{ds\_prepare\_gen\_c}(w_1, \mu)$
if $\texttt{calc\_seed} = \texttt{seed}$ **then**
  return OK
else
  return ERROR
end if

Як підсумок порівняльного аналізу, та враховуючи що схема підпису «Вершина» є Dilithium-подібною схемою цифрового підпису, яка ґрунтується на решітках, обидві реалізують модифіковану схему Fiat–Shamir з відсіченням (Fiat–Shamir with Rejection Sampling), можемо виокремити наступні результати щодо концептуальних, конструкційних та практичних відмінностей.

1. Параметри безпеки та режими роботи. `Dilithium` передбачає три параметричних набори, що відповідають класичним рівням безпеки 128, 192 та 256 біт.

Схема підпису «Вершина» розширює діапазон параметрів, підтримуючи чотири рівні безпеки: 128, 256, 384 та 512 біт. Останній ($\lambda = 512$) не реалізований у `Dilithium`, що робить схему підпису «Вершину» більш гнучкою у застосуваннях із підвищеними вимогами до безпеки.

2. Створення ключової пари. Обидві схеми використовують модульні решітки $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ та побудову матриці A, яка створюється випадковим чином з рядка ρ.

Схема підпису «Вершина» формує пари векторів s_1, s_2 та вектор t_0, який зберігається у відкритому ключі у скороченому вигляді t_1, через операцію усічення.

Схема підпису Dilithium також зберігає відкритий ключ у вигляді (ρ, t_1), проте використання відсічення для t дещо відрізняється.

3. Обчислення підпису. У схемі підпису «Вершина» використовується складніший підхід з багатоступеневою перевіркою норм різних векторів (z, r_0, $c \cdot t_0$), а також обчисленням матриці h для збереження допоміжної інформації. Це потребує кількох ітерацій циклу через процедуру відсічення.

У `Dilithium` підпис формується подібно, однак частина обчислень здійснюється в NTT-поданні, а перевірка зосереджена на векторі z та

поліномі c без додаткових відновлень r_0 або $c \cdot t_0$.

4. Маскування та захист від побічних каналів. Схема підпису `Вершина` чітко виокремлює компоненти маскування (y, r, z, $c \cdot t_0$) та виконує перевірку кожного з них на відповідність нормам. Це підвищує стійкість до атак побічних каналів, за умови правильної реалізації (наприклад, з постійним часом). Однак це все ще не є гарним підходом, оскільки ці секретні дані зберігаються у відкритому вигляді.

`Dilithium` менш орієнтований на індивідуальне маскування, однак зменшує ризик витоку інформації завдяки обмеженій структурі та простішій логіці перевірки.

5. Параметри підпису та ефективність. Схема підпису «Вершина» визначає конкретні розміри підпису та ключів через параметри `PK_OCTETS`, `SK_OCTETS`, `DS_OCTETS`, що залежать від вибраного режиму *mode*. Це дозволяє забезпечити високу криптографічну стійкість і водночас адаптивність.

`Dilithium` оптимізована для коротших підписів, але не підтримує варіант $\lambda = 512$, що обмежує її застосування в найвибагливіших середовищах.

7. Гнучкість використання. схеми підпису «Вершина» завдяки параметру *mode* дозволяє точно налаштовувати баланс між ефективністю та безпекою. Це робить її привабливою для широкого спектра застосувань: від вбудованих пристроїв до високозахищених систем.

Схема підпису `Dilithium` пропонує обмежену гнучкість, натомість забезпечуючи високу ефективність і простоту реалізації.

2.2 Модифікація схеми цифрового підпису «Вершина» з метою захисту від атак за побічним каналом

Схема цифрового підпису Dilithium, яка є переможцем в конкурсі NIST для постквантових цифрових підписів, продемонструвала вразливості до атак за побічним каналом. Зокрема, дослідження [15] показало, що використання аналізу споживання енергії дозволяє відновити значну частину секретного ключа, аналізуючи процедуру розпакування ключа під час підписування. Інше дослідження [16] продемонструвало, що атаки на процедуру відхилення викликів (rejection sampling) можуть призвести до повного відновлення секретного ключа за допомогою лінійного програмування. Ці атаки підкреслюють необхідність впровадження ефективних механізмів захисту від SCA в реалізаціях Dilithium. Враховуючи те, що схема цифрового підпису «Вершина» є Dilithium-подібною схемою, доцільно провести модифікацію даної схеми та спробувати підвищити стійкість до атак за побічним каналом.

На відміну від схеми підпису «Вершина», схема Raccoon була розроблена з урахуванням стійкості до атак за побічним каналом. Вона використовує техніку маскування, зокрема функцію `AddRepNoise`, яка забезпечує додавання повторного шуму для маскування критичних змінних під час виконання алгоритмів створення ключів та підписування. Це дозволяє обмежити витік інформації навіть у випадку, коли зловмисник має доступ до частини внутрішніх змінних під час виконання алгоритму. Дослідження [13] підтверджує ефективність цього підходу в моделі випадкового спостереження (probing model), де зловмисник може спостерігати обмежену кількість внутрішніх змінних.

Беручи до уваги виявлені вразливості в Dilithium та схемі підпису «Вершина» та ефективність захисних механізмів у Raccoon, виникає необхідність адаптації подібних підходів для схеми «Вершина». Зокрема, слід впровадити механізми маскування критичних змінних, таких як

секретні ключі та проміжні результати обчислень, з використанням технік, подібних до `AddRepNoise`. Це дозволить підвищити стійкість схеми до атак за побічним каналом, забезпечуючи надійність її реалізації в практичних умовах.

Якщо говорити більш конкретно, то основні проблемні місця в схемі підпису «Вершина» є:

1) **Зберігання секретних векторів s_1, s_2 у відкритому вигляді в пам'яті:** у схемі підпису «Вершина» ці вектори безпосередньо використовуються під час обчислення підпису у формулі $z = y + s_1 \cdot c$. Присутність s_1 у відкритому вигляді робить її надзвичайно вразливою до атак, заснованих на аналізі потужності або спостереженні регістрів, оскільки s_1 є головною секретною частиною секретного ключа.

2) **Обчислення добутку $s_1 \cdot c$:** під час обчислення $z = y + s_1 \cdot c$, відбувається доступ до елементів s_1 та c на рівні апаратного забезпечення. Оскільки c — відоме значення, кожне зчитування s_1 можна теоретично пов'язати з відповідним виходом z, що відкриває шлях до атаки через множинні трасування або навіть однотрасові атаки (як доведено в [15]).

3) **Відсутність маскування у векторах $t = A \cdot s_1 + s_2$:** вектор t або його округлені частини t_0, t_1 використовуються у публічному ключі та при перевірці підпису. Якщо в реалізації не передбачено захисту від побічного спостереження під час його обчислення, можливий витік інформації про s_1 через атаки на обчислення $A \cdot s_1$ (наприклад, через вимірювання часу або енергоспоживання в залежності від значень коефіцієнтів).

4) **Відсутність повторного додавання шуму до критичних змінних:** у схемі підпису «Вершина» значення s_1, s_2, а також результати z та t не підлягають жодному активному маскуванню чи повторному зашумленню. Це контрастує зі схемою Raccoon, де до всіх важливих змінних додається шум кілька разів за допомогою спеціалізованої функції `AddRepNoise`, що ускладнює атаки навіть при наявності доступу до частини обчислень.

5) **Наявність процедури відбору з відсіченнями** (`rejection`

`sampling`) **без приховування умов:** відхилення підпису на основі перевищення норми z може бути використане як оракул (oracle), оскільки на практиці процедура перевірки норми реалізується через умовні переходи, які можна інструментально спостерігати (через аналіз кешу або часу виконання), як показано в [16].

Таким чином, безпосередній доступ до секретних значень під час виконання ключових операцій підпису — найкритичніша вразливість Dilithium-подібних схем з точки зору атак за побічним каналом.

Підсумовуючи, схема цифрового підпису «Вершина» ґрунтується на схемі Dilithium, що каже нам про подібну структуру обчислень, зокрема операції з векторами s_1, s_2, y, z та множенням матриці A на вектори. Як показано у роботі [13], така структура є вразливою до атак за побічним каналом.

Враховуючи це, актуальним є покращення схеми цифрового підпису «Вершина» шляхом модифікації її алгоритмів **створення ключової пари** (KeyGen) та **підпису** (Sign), відповідно до змін, запропонованих у схемі **Raccoon**. Основна ідея полягає у тому, щоби зробити реалізацію стійкою до витоку значень s_1, s_2 та z під час обчислень.

У Raccoon це досягається за допомогою спеціалізованої функції `AddRepNoise`[12], яка додає шум до секретних компонентів кілька разів та ніколи не зберігає їх у чистому вигляді в пам'яті.

Нижчє наведено псевдокод функції, яка дозволяє додати шум

Аналізуючи псевдокод алгоритму 2.5, можемо виокремити дві ключові точки, у яких необхідно реалізувати маскування секретних векторів шляхом додавання повторного шуму за допомогою функції `AddRepNoise`:

– Після **рядка 6**, де відбувається ініціалізація векторів s_1, s_2 за допомогою виклику `ds_expand_s`. На цьому етапі потрібно додати:

$$s_1 \leftarrow \texttt{AddRepNoise}(s_1, D_\chi, r), \quad s_2 \leftarrow \texttt{AddRepNoise}(s_2, D_\chi, r)$$

Алгоритм 2.4 Функція `AddRepNoise` [12] для додавання повторного шуму

Input: Маскований вектор $[\![\mathbf{v}]\!] = (v_{i,j})_{i\in[\text{len}(\mathbf{v})],j\in[d]}$
Input: Параметр розподілу u (кількість бітів)
Input: Глобальний параметр повторень `rep`
Output: Оновлений маскований вектор $[\![\mathbf{v}]\!]$ з доданим шумом $SU(u, d \cdot \texttt{rep})$
for $i \in [\text{len}(\mathbf{v})]$ **do** ▷ Індекс вектора
  for $i_{\texttt{rep}} \in [\texttt{rep}]$ **do** ▷ Індекс повторення
    for $j \in [d]$ **do** ▷ Індекс частки
      $\sigma \leftarrow \{0,1\}^{\kappa}$ ▷ Секретний матеріал (випадковий вектор)
      $hdr_u \leftarrow \texttt{Ser}_{28}(117, i_{\texttt{rep}}, i, j, 0, 0, 0, 0)$
      $v_{i,j} \leftarrow v_{i,j} + \texttt{SampleU}(hdr_u, \sigma, u)$ ▷ Додати шум
    end for
    $[\![\mathbf{v}_i]\!] \leftarrow \texttt{Refresh}([\![\mathbf{v}_i]\!])$ ▷ Оновити масковану частину
  end for
end for
return $[\![\mathbf{v}]\!]$

Алгоритм 2.5 Псевдокод для створення ключової пари у схемі підпису «Вершина»[17]

Створення контексту:
  $ctx \leftarrow \texttt{AllocContext()}$
Створення матриці $A \in \mathcal{R}_q^{k\times l}$**:**
  $A \leftarrow \texttt{ds\_expand\_matr}(p, ctx)$
Створення векторів s_1**,** s_2**:**
  $s_1, s_2 \leftarrow \texttt{ds\_expand\_s}(p, ctx)$
Звільнення контексту:
  $\texttt{FreeContext}(ctx)$
Обчислення вектора $t := A \cdot s_1 + s_2$**:**
  $s_1 \leftarrow \texttt{vec\_ntt}(s_1, l)$
  $t \leftarrow \texttt{matr\_vec\_mul}(A, s_1)$
  $t \leftarrow \texttt{vec\_ntt}(t, k)$
  $t \leftarrow \texttt{vec\_add}(t, s_2, k)$
Обчислення t_0**,** t_1 **згідно формули 6.1:**
  $t_0[i], t_1[i] \leftarrow \texttt{ds\_gen\_hl}(t[i], 2^d)$
Упаковка відкритого ключа:
  $pk, pk\_len \leftarrow \texttt{ds\_pack\_pk}(p, t_1)$
Обчислення гешу відкритого ключа:
  $tr \leftarrow \texttt{Hash}(pk, pk\_len)$
Формування секретного ключа:
  $sk \leftarrow \texttt{ds\_pack\_sk}(p, key, tr, s_1, s_2, t_0)$

що забезпечить захист векторів до їх використання у множенні $A \cdot s_1 + s_2$ (рядок 9).

– Після **рядка 15**, де обчислюється представлення t_0, t_1 з вектора t. У випадку, якщо t буде частково збережений у пам'яті, також можливо доречно застосувати маскування до нього або безпосередньо до t_0 перед упаковкою ключа. Проте основним фокусом захисту залишається s_1, s_2, які потребують маскування одразу після створення.

Таким чином, інтеграція `AddRepNoise` у ці ключові точки дозволяє знизити ризик компрометації секретного ключа при реалізації підпису «Вершина» в апаратному середовищі.

Підкреслюючи ще раз, схема цифрового підпису «Вершина»[17] є Dilithium-подібною схемою цифрового підпису, тобто в алгоритмі підпису також використовується схема Фіат-Шаміра з відхиленнями [1]. Це означає, що з точки зору атак за побічним каналом (SCA) існує потенційна вразливість до витоку секретних параметрів під час обчислень. Особливо критичним є етап відбору з відсіченнями (rejection sampling), який створює залежність між поведінкою алгоритму та секретними даними. Тому логічно спробувати підвищити стійкість секретних параметрів, а також позбавитись від відбору з відсіченням.

Далі наведено рекомендації до того, яким чином можна покращити алгоритм підпису у схемі цифрового підпису «Вершина»

1 **Розпакування секретного ключа**
ρ, key, tr, s_1, s_2, t_0 := ds_unpack_sk(sk)

2 **Обчислення NTT-формату для s_1, s_2, t_0**
s_1 := vec_ntt(s_1, l);
s_2 := vec_ntt(s_2, k);
t_0 := vec_ntt(t_0, k);

3 **Формування буфера та та обчислення гешу повідомлення**
μ := Hash(tr $\|\|\|$ m)

4 **Відновлення матриці A згідно ρ**

$A :=$ ds_expand_matr(ρ);

5 **Обчислення вектора r та додавання випадкового шуму**

$r :=$ ZeroEncoding(d)

$r :=$ AddRepNoise(r, u_w, rep)

6 **Обчислення гешу та ініціалізація ГПВЧ для y**

buffer := key $|||$ μ

$ro_1 :=$ Hash(buffer)

ctx_y := PsevdoRandomInit(ctx_y, ro_1, HASH_OCTETS)

7 **Обчислення вектора y та перетворення у NTT**

y, ctx_y := ds_gen_y(ctx_y);

ntt_y := vec_ntt(y, l);

8 **Обчислення $w = A \cdot y$**

ntt_w := matr_vec_mul(A, ntt_y);

w := vec_ntt(ntt_w)

w := AddRepNoise(w, u_w, rep)

9 **Обчислення c_{hash} та c_{poly}**

w1[i], w0[i] := ds_gen_hl(w[i], 2 $\cdot\gamma_2$) $(i = 0,1,...,k-1)$

seed := ds_prepare_gen_c(w1, μ);

$c_{hash} :=$ `ChalHash`(w, μ) [12]

$c_{poly} :=$ `ChalPoly`(c_{hash}) [12]

10 **Обчислення $z = y + c \cdot s_1$**

z[i] := pointwise(c_{poly}, s1[i]) $(i = 0, ..., l-1)$

z := vec_ntt(z, l)

11 **Обчислення допоміжного вектора та гешу h**

r[i] := pointwise(c_{hash}, s2[i]);

r := vec_ntt(r, k);

r := vec_sub(w, r, k);

ct0[i] := pointwise(c', t0[i]);

ct0 := vec_ntt(ct0, k);

temp := vec_add(r, ct0, k);

temp1[i], temp0[i] := ds_gen_hl(temp[i], $2{\cdot}\gamma_2$)

h, cnt := ds_gen_h(c_{hash}, temp1);

12 **Формування підпису та перевірка норми**

ds := ds_pack_ds(c_{hash}, z, h)

if CheckBounds(ds) = ERROR **then** goto 5 **end if**

Алгоритм 2.6 Перевірка коректності підпису `CheckBounds`(sig) [12]

Вхід: серіалізований підпис $sig = (c_{\text{hash}}, \mathbf{h}, \mathbf{z})$
Вихід: `OK` або `FAIL`
if $|sig| \neq |sig|_{\text{default}}$ **then**
    return `FAIL` ▷ Невірна довжина або кодування підпису
end if
Розпакувати sig: $(c_{\text{hash}}, \mathbf{h}, \mathbf{z}) \leftarrow sig$
if $\|\mathbf{h}\|_\infty > \lfloor B_\infty / 2^w \rfloor$ **then**
    return `FAIL` ▷ Вихід за межу по **h**
end if
if $\|\mathbf{z}\|_\infty > B_\infty$ **then**
    return `FAIL` ▷ Вихід за межу по **z**
end if
$h_2 \leftarrow 2^{(2v_w - 64)} \cdot \|\mathbf{h}\|_2^2$ ▷ Масштабована L_2-норма для **h**
$z_2 \leftarrow \left(\sum_i \left\lfloor \frac{|\mathbf{z}_i|}{2^{32}} \right\rfloor^2 \right)$ ▷ Масштабована L_2-норма для **z**
if $h_2 + z_2 > 2^{-64} \cdot B_2^2$ **then**
    return `FAIL` ▷ Перевищення комбінованої евклідової норми
end if
return `OK`

Основними покращеннями в даному псевдокоді є перш за все додавання **AddRepNoise** на 5 та 8 кроці відповідно, тобто напряму додаємо випадковий шум в секретні параметри, окрім цього прибирається пряме зберігання змінної c та додаються дві нових змінних c_{hash} та c_{poly}, а також прибрана процедура відбору з відсіченнями (rejection sampling) з 12 кроку та замінений на алгоритм **CheckBounds**[12]. Варто зазначити, що необхідності в маскуванні підпису вже немає, тому цей крок це проста перевірка норми.

Однак тепер постає запитання, чи додані зміни не погіршили

криптографічну стійкість схеми, тому доцільно розглянути питання безпеки модифікованого підпису.

Означення 2.1 (Схема цифрового підпису «Вершина»[17])**.** Схема складається з трьох основних алгоритмів:

– **KeyGen**: Створює ключову пару (pk, sk);

– **Sign**: Створює підпис повідомлення m за допомогою секретного ключа sk;

– **Verify**: Перевіряє коректність підпису σ для повідомлення m.

Підпис має вигляд:

$$\sigma := (c, z, h), \quad \text{де} \quad z = y + c \cdot s_1, \quad h^{\smile} \text{допоміжні дані.}$$

Ключові зміни, запроваджені для захисту від атак за побічними каналами, включають використання функції повторного додавання шуму (AddRepNoise) та заміна етапу відбору з відсіченням на перевірку норми використовуючи функцію CheckBounds.

Означення 2.2 (Модульна задача навчання на помилках (MLWE))**.** Нехай ℓ, k, q — цілі числа, а $\mathcal{D}$ — розподіл імовірностей над R_q. Перевага зловмисника $\mathcal{A}$ у задачі модульного навчання з шумами ($\mathrm{MLWE}_{q,\ell,k,\mathcal{D}}$) визначається як:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{MLWE}}(\kappa) = \left|\Pr\left[1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})\right] - \Pr\left[1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{b})\right]\right|,$$

де $(\mathbf{A}, \mathbf{b}, \mathbf{s}, \mathbf{e}) \leftarrow R_q^{k \times \ell} \times R_q^k \times \mathcal{D}^\ell \times \mathcal{D}^k$.

Припущення $\mathrm{MLWE}_{q,\ell,k,\mathcal{D}}$ стверджує, що будь-який поліноміально обмежений зловмисник $\mathcal{A}$ має нехтовно малу перевагу.

Означення 2.3 (Модульна задача короткого цілочисельного розв'язку (MSIS))**.** Нехай ℓ, k, q — цілі числа та $\beta > 0$ — дійсне число. Перевага зловмисника $\mathcal{A}$ у задачі модульного короткого цілочисельного розв'язку ($\mathrm{MSIS}_{q,\ell,k,\beta}$) визначається як:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{MSIS}}(\kappa) = \Pr\left[\mathbf{A} \leftarrow R_q^{k \times \ell}, \mathbf{s} \leftarrow \mathcal{A}(\mathbf{A}) : \ 0 < \|\mathbf{s}\|_2 \leqslant \beta \ \wedge \ [\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s} \equiv 0 \pmod{q}\right.$$

Припущення $\mathrm{MSIS}_{q,\ell,k,\beta}$ стверджує, що будь-який ефективний зловмисник $\mathcal{A}$ має нехтовну перевагу.

Однак в даній роботі та наступній теоремі більше буде використана модифікована задача `MSIS`.

Означення 2.4 (Задача SelfTargetMSIS)**.** [9, 2] Нехай $H : \{0,1\}^* \to \mathcal{B}_{60}$ — криптографічна геш-функція. Для алгоритму $\mathcal{A}$ визначимо функцію переваги в задачі SelfTargetMSIS наступним чином:

$$\mathrm{Adv}^{\mathrm{SelfTargetMSIS}}_{H,m,k,\gamma}(\mathcal{A}) := \Pr\left[\begin{array}{l|l} & \mathbf{A} \leftarrow R_q^{m\times k}; \\ 0 \leqslant \|\mathbf{y}\|_\infty \leqslant \gamma \wedge & (\mathbf{y}, M) \leftarrow \mathcal{A}(H^{(\cdot)}(\mathbf{A})) \\ H([\mathbf{I} \mid \mathbf{A}] \cdot \mathbf{y} \parallel M) = c & \mathbf{y} := \begin{bmatrix} \mathbf{r} \\ \\ \mathbf{c} \end{bmatrix} \\ & \end{array}\right].$$

Припущення задачі SelfTargetMSIS стверджує, що для будь-якого поліноміального зловмисника $\mathcal{A}$ його перевага $\mathrm{Adv}^{\mathrm{SelfTargetMSIS}}_{H,m,k,\gamma}(\mathcal{A})$ є нехтовно малою.

Означення 2.5 (EUF-CMA безпека)**.** Схема $\mathcal{S}$ є EUF-CMA безпечною, якщо для будь-якого поліноміального зловмисника $\mathcal{A}$ виконується наступне співвідношення:

$$\mathrm{Adv}^{\text{EUF-CMA}}_{\mathcal{A}}(\mathcal{S}) \leqslant \mathrm{negl}(\kappa), \tag{2.1}$$

де $\mathrm{negl}(\kappa)$—функція, що прямує до нуля швидше будь-якого полінома від κ [8].

Теорема 2.1. *Схема підпису «Вершина» є EUF-CMA безпечною за припущенням складності задач Module-LWE та SelfTargetMSIS.*

Доведення. Проведемо спрощене доведення за допомогою серії ігор, аналогічно до підходу, використаному в схемі Raccoon[12].

Розглянемо зловмисника $\mathcal{A}$, який має доступ до оракула підпису:

Гра 0: Оригінальна схема підпису «Вершина».

Гра 1: Створення матриці A, за допомогою функції ds_expand_matr. За припущенням Module-LWE, створення випадкової матриці є непомітним для зловмисника.

Гра 2: Замість початкових s_1, s_2 використовується AddRepNoise:

$$s_1' = s_1 + e, \quad s_2' = s_2 + e',$$

де e, e'—незалежний шум з дискретного гаусівського розподілу з параметром α.

Гра 3: Значення c_{poly} та c_{hash}обчислюється незалежно від секретного ключа шляхом звертання до оракула. Невідрізнюваність гарантується складністю задачі SelfTargetMSIS[2], аналогічно до доведення в Raccoon [13].

Гра 4: Заміна обчислення $z = y + c \cdot s_1$ на незалежну вибірку за розподілом, що не залежить від s_1. Ця заміна можлива завдяки припущенню Module-LWE.

Гра 5: Етап відбору з відсіченнями замінюється перевіркою CheckBounds:

$$\|z\|_\infty \leqslant \gamma - \beta,$$

що виключає витік інформації через відмову, яка могла залежати від секретних ключів.

Аналіз ігор:

– Перехід між іграми 0→1 ґрунтується на Module-LWE.

– Перехід між іграми 1→2 не збільшує перевагу $\mathcal{A}$, завдяки властивостям AddRepNoise.

– Перехід між іграми 2→3 ґрунтується на SelfTargetMSIS.

– Перехід між іграми 3→4 ґрунтується на складності Module-LWE.

– Перехід 4→5 не впливає на безпеку, оскільки CheckBounds не розкриває інформацію.

Таким чином, перевага зловмисника обмежена:

$$\mathrm{Adv}_{\mathcal{A}}^{\text{EUF-CMA}} \leqslant \mathrm{Adv}_{\mathcal{B}}^{\text{Module-LWE}} + \mathrm{Adv}_{\mathcal{B}'}^{\text{SelfTargetMSIS}} + Q_s \cdot \varepsilon_{TAIL},$$

де Q_s— кількість запитів до оракула, ε_{TAIL}—ймовірність перевищення межі норми:

$$\varepsilon_{TAIL} = \Pr[\|z\|_\infty \geqslant \gamma - \beta].$$

Вибір параметрів гарантує, що ε_{TAIL} є нехтовно малою[13]. Аналогічні розрахунки підтверджують ефективність схеми для $\kappa = 512$ (в схемі підпису «Вершина» це λ), зберігаючи відповідну стійкість, яка є сильнішою за обрані параметри в Raccoon [12]. □

Варто також зазначити оцінку переваги зловмисника при застосуванні `AddRepNoise`[12].

Лема 2.1 (Вплив AddRepNoise[12])**.** *Для будь-якого поліноміально обмеженого зловмисника $\mathcal{A}$ використання функції AddRepNoise забезпечує наступну оцінку:*

$$Adv_{\mathcal{A}}^{Hybrid'} \leqslant \left(Adv_{\mathcal{A}}^{Hybrid}\right)^{\frac{\alpha-1}{\alpha}} \cdot \exp\left(o\left(\frac{1}{\kappa^{3/2}}\right)\right),$$

де $\alpha > 1$— параметр згладженої Реньї-розбіжності, κ— параметр безпеки схеми.

2.3 Оптимізація обчислювальної складності алгоритму підпису в схемі цифрового підпису «Вершина»

Доцільно розглянути можливість оптимізації схеми цифрового підпису «Вершина» шляхом інтеграції підходу, відомого як **оптимізація з використанням попередніх обчислень (early evaluation optimization)**[14], до процедури підпису `Sign`. Ідея оптимізації полягає у поетапному обчисленні та негайній перевірці кожного полінома у векторі

(наприклад, $z[i]$), до того як буде обчислений наступний. Такий підхід дозволяє уникнути зайвих обчислень, якщо вже на якомусь етапі вектор не проходить норму:

$$\text{if } \neg\mathsf{Chk\_Norm}(z[i]) \Rightarrow \text{відхиляємо ітерацію}$$

Це дозволяє зекономити обчислювальні ресурси, уникаючи непотрібних операцій над компонентами $z[i+1], \ldots, z[L-1]$. Варто зазначити, що в оригінальному описі схеми цифрового підпису «Вершина» [17] деталі реалізації перевірки норми не наведені, що відкриває можливість для уточнення або вдосконалення цієї частини алгоритму.

Алгоритм 2.7 Оптимізована функція `Sign` з раннього оцінкою

Input: повідомлення m, секретний ключ $sk = (\rho, key, tr, s_1, s_2, t_0)$
Output: підпис (z, c, h)
Початкова ініалізація алгоритму
repeat
    ...
    Перевірка норми для z: $\mathsf{Chk\_Norm}(z)$
    Перевірка норми для $r0$: $\mathsf{Chk\_Norm}(r0)$
    Перевірка норми для $ct0$: $\mathsf{Chk\_Norm}(ct0)$
    Перевірка ваги w: $\mathsf{Chk\_Weight}(w)$
    Перевіряємо, чи $r_1 \neq w_1$
until успішного завершення

У класичній реалізації Dilithium спочатку обчислюється повністю весь вектор $z = (z[0], z[1], \ldots, z[L-1])$, і лише потім виконується перевірка норми. У результаті, навіть якщо $z[0]$ не проходить перевірку, вся ітерація вже витратила ресурси на обчислення непотрібних $z[1], z[2], \ldots$ [14].

У випадку з схемою підпису «Вершина»[17] та посилаючись на **оптимізацію з використанням попередніх обчислень** [14] дана техніка вже частково було застосовна, однак не було перевірки рівності

$r_1 \neq w_1$, а також не наведено реалізації ds_check_vect.

Запропоновано реалізацію аналогічної *оптимізації з використанням попередніх обчислень* при обчисленні векторів z, r_0 та ct_0 у схемі «Вершина». Ми розбиваємо ланцюг обчислень на обчислення одного полінома за раз і одразу після цього виконуємо перевірку, за аналогією до прикладу з оригінального джерела[14]:

Алгоритм 2.8 Обчислення z з використанням попередніх обчислень

```
for i = 0 to L − 1 do
    z1[i] ← poly_pointwise_invmontgomery(s1[i], ĉ)
    z2[i] ← poly_invntt_montgomery(z1[i])
    z3[i] ← poly_add(y[i], z2[i])
    z[i] ← poly_freeze(z3[i])
    if ¬polyvec_chknorm(z[i], γ1 − β) then
        goto Repeat
    end if
end for
```

Підсумовуючи, дана оптимізація має кілька ключових переваг. По-перше, вона дозволяє усунути зайві обчислення: якщо на якомусь етапі поліном $z[i]$ не задовольняє умову норми, то вся ітерація негайно перезапускається, не витрачаючи ресурси на обчислення наступних поліномів $z[j]$. По-друге, реалізація такого підходу є простою, оскільки всі функції, що застосовуються до поліномів, такі як add, freeze та chknorm, є поелементними і не залежать від інших компонентів вектора. Нарешті, оптимізація не створює загроз з боку атак за побічним каналом: хоча вона й змінює загальний час виконання алгоритму, ймовірність відхилення окремого полінома не залежить від секретного ключа, а отже — не витікає жодної конфіденційної інформації.

Висновки до розділу 2

У цьому розділі проведено порівняльний аналіз схеми цифрового підпису «Вершина» та еталонної схеми підпису Dilithium, що дозволило виявити ключові відмінності у параметрах, структурі алгоритмів та рівнях безпеки, а також визначити шляхи подальшого удосконалення.

Для підвищення стійкості до атак за побічним каналом у схемі підпису «Вершина» було впроваджено низку модифікацій. Зокрема, додано випадковий шум до секретних параметрів як на етапі створення ключів, так і під час обчислення підпису. Це суттєво ускладнює витік інформації через фізичні характеристики пристроїв. Окрім цього, усунено механізми помилкових відмов (false rejections), що усуває потенційні витоки інформації через залежність поведінки алгоритму від секретних даних.

Також проведено оптимізацію обчислювальної складності алгоритму підпису шляхом використання стратегії передчасного обчислення(тобто виконання певних обчислень наперед, до появи критичних розгалужень), що дозволило зменшити витрати ресурсів без погіршення безпеки.

Важливим результатом є доведення теореми щодо коректності та безпечності модифікованої схеми підпису «Вершина». У роботі наведено логічні обґрунтування, які демонструють, що запропоновані зміни не впливають негативно на основні властивості схеми та не знижують рівень криптографічної стійкості.

Загалом, результати цього розділу засвідчують ефективність запропонованих підходів до підвищення захищеності й ефективності схеми підпису «Вершина» у порівнянні з класичними реалізаціями.

ВИСНОВКИ

У результаті проведеного дослідження було виконано глибокий аналіз сучасних постквантових схем цифрового підпису, зосереджений на порівнянні конструкцій, принципів роботи та типових вразливостей, зокрема атак за побічним каналом. Особливу увагу приділено схемі підпису «Вершина», для якої визначено потенційні місця для вдосконалення щодо підвищення стійкості та ефективності.

Запропоновані в роботі модифікації полягають у впровадженні випадкового шуму до секретних параметрів на етапах створення ключів і обчислення підпису, що дозволяє суттєво ускладнити спроби компрометації секретної інформації через аналіз фізичних характеристик обчислювальних пристроїв. Усунено механізм помилкових відмов (false rejections), що виключає витік інформації через залежність поведінки алгоритму від вмісту секретних даних. Впроваджена оптимізація на основі передчасного обчислення дозволила зменшити обчислювальні витрати та підвищити ефективність процесу підписування без втрати рівня криптографічної стійкості.

Показано доведення того, що запропоновані зміни не призводять до зниження безпеки схеми. Проведений порівняльний аналіз із класичними схемами підпису, зокрема Dilithium, підтвердив можливість досягнення гнучких налаштувань під конкретні вимоги користувачів, включаючи підтримку високого рівня криптостійкості.

У перспективі подальші дослідження можуть бути зосереджені на експериментальній перевірці ефективності запропонованих оптимізацій з погляду обчислювальної складності, а також на практичній перевірці стійкості модифікованої схеми підпису «Вершина» до реальних атак за побічним каналом на різних апаратних платформах.

ПЕРЕЛІК ПОСИЛАНЬ

[1] Julien Devevey та ін. *A Detailed Analysis of Fiat-Shamir with Aborts*. Трав. 2023. URL: https://eprint.iacr.org/2023/245/20240514:160408.

[2] L. Ducas та ін. *CRYSTALS-Dilithium: A lattice-based digital signature scheme*. Т. 2018. 1. 2018, с. 238—268. URL: https://tches.iacr.org/index.php/TCHES/article/view/839.

[3] Léo Ducas, Thomas Espitau та Eamonn W. Postlethwaite. *Finding short integer solutions when the modulus is small*. Лип. 2023. URL: https://eprint.iacr.org/2023/1125/20230719:140814.

[4] T. Espitau та ін. *MITAKA: A Simpler, Parallelizable, Maskable Variant of FALCON*. hal-03627833. Trondheim, Norway, 2022, с. 1—50.

[5] P.-A. Fouque та ін. *Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. Specification v1.2*. Жовт. 2020. URL: https://falcon-sign.info/falcon.pdf.

[6] C. Gentry, C. Peikert та V. Vaikuntanathan. *Trapdoors for hard lattices and new cryptographic constructions*. Victoria, BC, Canada: ACM Press, 2008, с. 197—206.

[7] J. Hoffstein та ін. *NTRUSIGN: Digital signatures using the NTRU lattice*. За ред. M. Joye. Т. 2612. LNCS. Springer, 2003, с. 122—140.

[8] J. Katz та Y. Lindell. *Introduction to Modern Cryptography*. 3-е вид. Chapman та Hall/CRC, 2020.

[9] Eike Kiltz, Vadim Lyubashevsky та Christian Schaffner. *A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model*. Springer International Publishing, 2018, с. 552—586. ISBN: 9783319783727. DOI: 10.1007/978-3-319-78372-7_18.

[10] Alessio Meneghetti та Edoardo Signorini. *History-Free Sequential Aggregation of Hash-and-Sign Signatures*. Черв. 2023. DOI: 10 . 1007 / 978 - 3 - 031 - 58868 - 6 _ 8. URL: https://eprint.iacr.org/2023/784/20240625:120804.

[11] J. Noh, D. Han та D.-J. Shin. *Efficient Error-tolerant Side-channel Attacks on GPV Signatures Based on Ordinary Least Squares Regression*.

[12] R. del Pino та ін. *A Side-Channel Secure Signature Scheme*. 2024.

[13] R. del Pino та ін. *Raccoon: A Masking-Friendly Signature Proven in the Probing Model*. T. 2024. 2024, с. 1291.

[14] P. Ravi та ін. *Improving Speed of Dilithium's Signing Procedure*. NTU, Singapore. 2024.

[15] R. Wang та ін. *Single-Trace Side-Channel Attacks on CRYSTALS-Dilithium: Myth or Reality?* T. 2023. 2023, с. 1931.

[16] Y. Zhou та ін. *Rejected Challenges Pose New Challenges: Key Recovery of CRYSTALS-Dilithium via Side-Channel Attacks*. T. 2025. 2025, с. 214.

[17] *Інформаційні технології: Криптографічний захист інформації. Алгоритми електронного підпису на алгебраїчних решітках з відхиленнями.*