Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)
Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Дослідження постквантумних алгоритмів. Цифровий підпис
(тема)

Виконав:
Студент 2 курсу групи ІПЗм-22-2
Неділько Д.С
(прізвище, ініціали)

Спеціальність: 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми: освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Керівник: професор Власенко Л.А.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. Кафедри _____ Дудар З.В.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

Рівень вищої освіти другий магістерський

Спеціальність 121-Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ

Зав. кафедри _____
(підпис)
«_____» _____ 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Неділько Дмитро Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи: Дослідження постквантумних алгоритмів. Цифровий підпис затверджена наказом по Університету від «29» березня 2024 року № 250 Ст.

2. Термін подання студентом роботи до екзаменаційної комісії 25 червня 2024р.

3. Вихідні дані до роботи: алгоритми схем підписів, математичні методи протоколів цифрових підписів, дані проектів щодо розробки сучасних систем підписів, мова Python, схеми багаторазових підписів на деревах, алгоритми обходу, параметри ефективних схем підписів.

4. Перелік питань, що потрібно опрацювати в роботі:

1)Аналіз методів постквантової криптографії

2)Аналіз постквантових систем підписів

3)Визначення та аналіз алгоритмів побудови сучасних систем підписів

4)Аналіз схем підписів, порівння алгоритмів підписів, моделювання та порівняльний аналіз.

4) Моделювання OTS Лампорта та схеми підпису на дереві Меркла

5) Вивчення проблеми обходу дерева Меркла для обчислення ефективного шляху автентифікації

6) Дослідження з метою вибору найбільш збалансованих значень параметрів.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1	Збір матеріалів для дослідження	01.04.2024	Виконано
2	Аналіз предметної галузі	10.04.2024	Виконано
3	Дослідження існуючих алгоритмів постквантових підписів	15.04.2024	Виконано
4	Аналіз моделі, моделювання	17.04.2024	Виконано
5	Аналіз результатів моделювання	20.04.2024	Виконано
6	Написання пояснювальної записки	10.05.2024	Виконано
7	Перевірка пояснювальної записки	01.06.2024	Виконано
8	Оцінка роботи рецензентом	18.06.2024	Виконано
9	Здача роботи у електронний архів	22.06.2024	Виконано
10	Попередній захист	23.06.2024	Виконано
11	Допуск до захисту у завідувача кафедри	24.06.2024	Виконано
12	Захист кваліфікаційної роботи	25.06.2024	Виконано

Дата видачі завдання 1 квітня 2024 р.

Студент  (підпис)

Керівник роботи _____ (підпис)

РЕФЕРАТ/ ABSTRACT

Пояснювальна записка: 86 с., 9 рис., 5 табл., 6 додатка, 28 джерел

ПОСТКВАНТОВА КРИПТОГРАФІЯ, ЕЛЕКТРОННИЙ ЦИФРОВИЙ ПІДПИС, СХЕМА ПІДПИСУ, ХЕШ, HBS, СХЕМА ЛАМПОРТА, ДЕРЕВА МЕРКЛА, СХЕМА ВІНТЕРНІЦА, SPRINGS.

Об’єкт дослідження – процеси в постквантових системах підпису.

Предмет дослідження – методи і засоби створення сучасних систем підпису.

Мета роботи – аналіз шляхів підвищення ефективності побудови сучасних систем підпису, які стійкі до квантових алгоритмів.

Методи дослідження – аналіз схем підписів, порівняння алгоритмів підписів, моделювання та порівняльний аналіз.

У роботі проведено аналіз шляхів підвищення ефективності побудови сучасних постквантових алгоритмів цифрових підписів. Представлено огляд різних напрямів досліджень, які вивчаються в постквантовій криптографії. Розглянуто схеми цифрового підпису, проведено порівняльний аналіз постквантових алгоритмів. Проведено аналіз різноманітних схем підписів на основі хеша. Розглянуто алгоритм OTS Лампорта. Детально вивчені схеми багаторазових підписів, які реалізовано на деревах Меркла. Розроблена робоча модель ОТС Лампорта, на основі цього моделювалося дерево Меркла для отримання багаторазових підписів. Вивчалася проблема обходу дерева Меркла для обчислення ефективного шляху автентифікації. Проведен аналіз подібності та відмінності між алгоритмами підпису, було розглянуто вимоги до часу та простору для кожної техніки обходу з порівнянням їх ефективність у різних ситуаціях. Проведено дослідження з метою вибору найбільш збалансованих значень параметрів.

POST-QUANTUM CRYPTOGRAPHY, ELECTRONIC DIGITAL SIGNATURE, SIGNATURE SCHEME, HASH, HBS, LAMPORT SCHEME, MERKLE TREE, WINTERNITZ SCHEME, SPRINGS.

The object of research is processes in post-quantum writing systems.

The subject of research is methods and means of creating modern signature systems.

The purpose of the work is to analyze ways to increase the efficiency of building modern signature systems that are resistant to quantum algorithms.

Research methods – analysis of signature schemes, comparison of signature algorithms, modeling and comparative analysis.

Ways to improve the efficiency of constructing modern post-quantum digital signature algorithms are analyzed. An overview of various research areas studied in post-quantum cryptography is presented. Digital signature schemes are considered, and a comparative analysis of post-quantum algorithms is carried out. An analysis of various hash-based signature schemes was carried out. Lamport's OTS algorithm is considered. The schemes of multiple signatures, which are implemented on Merkle trees, are studied in detail. A working model of Lamport's OTS was developed, on the basis of which a Merkle tree was modeled for obtaining multiple signatures. The Merkle tree traversal problem was studied to calculate an efficient authentication path. An analysis of the similarities and differences between the signature algorithms was conducted, and the time and space requirements for each bypass technique were considered, with a comparison of their effectiveness in different situations. Research was conducted to select the most balanced parameter values.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Неділько Дмитро Сергійович, студент гр. ІПЗм-22-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження постквантумних алгоритмів. Цифровий підпис.", що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

PQC (Post Quantum Cryptography) – постквантова криптографія;

TLS (Transport Layer Security) – протокол безпеки транспортного рівня;

DNS (Domain Name System) – система доменних імен;

RSA (абревіатура від прізвищ Rivest, Shamir та Adleman) — криптографічний алгоритм з відкритим ключем;

DSA (Digital Signature Algorithm) – криптографічний алгоритм з використанням відкритого ключа для створення електронного підпису;

MSS (Merkle Signature Scheme) – це схема цифрового підпису, заснована на хеш-деревах (дерева Меркла) і одноразових підписах, таких як схема підпису Лампорта;

NIST (National Institute of Standards and Technology) – Національний інститут стандартів і технологій;

SPHINCS (stateless hash-based signatures) – криптографічна схема, заснована на криптографічній стійкості хеш-функції;

HFE (Hidden Field Equations) – приховані рівняння поля (різновид криптографічної системи з відкритим ключем, що є частиною багатовимірної криптографії);

UOV (Unbalanced Oil and Vinegar) – багатовимірна криптографічна схема, використовується для підписів;

CRYSTAL (Cryptographic Suite for Algebraic Lattices) – охоплює набір криптографічних алгоритмів, що базуються на проблемах гратки;

HBS (hash-based signature) – підпис на основі хешу;

OTS (one-time signature) – одноразовий підпис;

WOTS (Winternitz one-time signature) – одноразовий підпис Вінтерніца;

HORS (Hash to Obtain Random Subset) – хеш для отримання випадкової підмножини

ВСТУП

У розмові віч-на-віч ми часто можемо засвідчити особу людини, знаючи та перевіряючи її обличчя, голос або хтось інший представляє її нам. У цифровому світі підтвердити свою особу складніше. Коли користувач підключається до свого банківського провайдера в Інтернеті, йому потрібна впевненість не тільки в тому, що інформація, яку він надсилає, захищена; але вони також надсилають його до свого банку, а не до шкідливого веб-сайту, який маскується під їх постачальника. Протокол безпеки транспортного рівня (TLS) забезпечує це за допомогою цифрових підписів про ідентифікацію (сертифікатів). Схеми цифрового підпису також відіграють центральну роль у DNSSEC, розширенні системи доменних імен (DNS), яка захищає програми від прийняття підроблених або маніпуляційних даних DNS, що відбувається, наприклад, під час отруєння кешу DNS.

Криптографія використовувалася з незапам'ятних часів для збереження конфіденційності даних/інформації під час зберігання або передачі. Дослідження криптографії також еволюціонували від класичного шифру Цезаря до сучасних криптосистем, заснованих на модульній арифметиці [1], до криптосистем, заснованих на квантових обчисленнях. Поява квантових обчислень створює серйозну загрозу для сучасних криптосистем, заснованих на модульній арифметиці, завдяки чому навіть складні обчислювальні проблеми, які становлять міцність модульних арифметичних шифрів, можуть бути вирішені за поліноміальний час. Ця загроза спровокувала дослідження постквантової криптографії для розробки постквантових алгоритмів, здатних протистояти атакам квантових обчислень.

Потрібно зробити кілька концептуальних зауважень. Класична криптографія орієнтована те, щоб приховати інформацію від стороннього погляду. Це досягається підбором відповідної функції. Таким чином, це розділ математики. Проте є теоретична можливість зламати код і відновити значення цієї функції. Звичайно, це займе дуже тривалий час. У квантовій криптографії зламати код

неможливо. Взагалом вона відноситься не до чистої математики, а скоріше до побудови безпечних квантових систем передачі даних. Постквантова криптографія - це розвиток ідей класичної криптографії, на основі яких можна протистояти загрозам квантових комп'ютерів.

Кваліфікаційна робота апробована на XXVI INTERNATIONAL SCIENTIFIC AND PRACTICAL CONFERENCE «Theoretical and Practical Aspects of Modern Research» June 5-7, 2024 Ottawa, Canada.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Цей розділ містить огляд різних напрямів досліджень, які вивчаються в постквантовій криптографії, але перед цим розглядаються алгоритми та методи квантових обчислень, щоб зрозуміти, які загрози вони несуть безпеці криптографічних схем.

1.1 Квантові обчислення

Квантовий комп’ютер — це машина, яка використовує квантово-фізичні явища для виконання обчислень у спосіб, який принципово відрізняється від «нормального» класичного комп’ютера [2]. Тоді як класичний комп’ютер у будь-який момент часу перебуває у фіксованому стані — наприклад, бітовий рядок, який представляє вміст його пам’яті — стан квантового комп’ютера може бути «сумішшю», так званою суперпозицією, кількох станів. Класичні комп'ютери виконують логічні операції, використовуючи певне положення фізичного стану. Зазвичай вони двійкові, тобто їх операції базуються на одній із двох позицій.

Окремий стан, наприклад, увімкнено чи увімкнено, вгору чи вниз, 1 або 0, називається бітом. Зауважте, що внутрішній стан прихований: єдиний спосіб отримати інформацію про стан — це виконати вимірювання, яке поверне єдиний класичний результат без накладення, наприклад бітовий рядок, випадково розподілений відповідно до внутрішнього стану, і внутрішній стан замінюється результатом вимірювання. Квантовий біт прийнято називати кубіт. На відміну від класичного біта кубіт може перебувати, як показано рисунку 1.1, у стані 0 ($a = 0$), 1 ($b = 0$) чи стані суперпозиції ($a \neq 0, \quad b \neq 0$). Зазначимо, що a і b є

комплексними числами, квадрат їх модуля дає ймовірність знаходження в стані після вимірювання. Виходячи з цього, сумарна ймовірність повинна дорівнювати одиниці: $|a|^2 + |b|^2 = 1$.

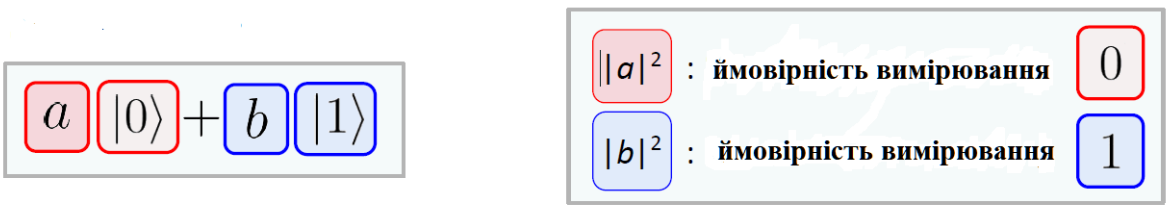

Рисунок 1.1 – Квантовий кубіт

В процесі обчислень проходять маніпуляції з кубітами. У квантових комп'ютерах обчислення це оборотні операції, на відміну від класичних, але до тих пір, поки не знімаємо показання про стан кубіта. Якщо взяти єдиний кубіт, не матимемо жодного виграшу в ефективності обчислень. Але тут слід зазначити, що в процесі обчислень відбувається еволюція двох станів, на відміну від єдиного у разі класичного процесора. Таким чином, для класичних змінних група n біт зберігає єдине з 2^n можливих значень на кожному етапі обчислень. Але група з n кубітів зберігає вже 2^n можливих значень із визначеною ймовірністю вимірювання кожного окремого стану. Нещодавно корпорація IBM представила перший модульний квантовий комп'ютер IBM Quantum System Two на базі новітнього 133-кубітного квантового процесора Heron. Таким чином, на кожному етапі вирахування відбувається перетворення 2^{133} станів. Вчені з Китаю повідомляють, що їх новий квантовий комп'ютер вирішив надзвичайно складне математичне завдання за мільйонну частку секунди. Це на 20 мільярдів років швидше, ніж міг би впоратися з таким самим завданням найшвидший у світі суперкомп'ютер. Але не все так оптимістично, оскільки необхідно створити ефективні алгоритми для роботи на квантових комп'ютерах. До того ж необхідно вирішити ряд фізичних проблем, пов'язаних із втратою когерентності,

виправленням помилок. Тому навіть такі вражаючі результати - це швидше моделювання фізичного процесу, ніж конкретні обчислення. У будь-якому випадку квантові обчислення є новою обчислювальною парадигмою, яка, як очікується, вирішуватиме складні проблеми, що вимагають набагато більшої обчислювальної потужності, ніж ті завдання, які можливо вирішити з поточним поколінням комп'ютерних технологій.

1.1.1 Квантові алгоритми

У розробці квантових алгоритмів є два новаторські алгоритми, які заклали міцну основу для зламу великої кількості просунутих криптосистем з відкритим ключем. У 1994 році Шор запропонував поліноміально-часовий (ефективний) алгоритм [3] для розв’язування задач цілочислової факторизації та дискретного логарифмування. Алгоритм ґрунтується на існуванні квантових комп’ютерів, тому цей тип алгоритмів у цій статті називається квантовими алгоритмами. Квантовий алгоритм Шора та його варіанти можна використовувати для зламу більшості використовуваних на даний момент криптосистем з відкритим ключем.

Алгоритм для знаходження дискретних логарифмів використовує модульне зведення в ступінь та квантову операцію, яка називається квантовим перетворенням Фур’є, щоб знайти дискретний логарифм за поліноміальний час [3]. Алгоритм факторизації використовує той факт, що факторизацію n можна звести до знаходження порядку елемента x у мультиплікативній групі (mod n). Іншими словами, треба знайти r таке, щоб $x^r \equiv 1 \bmod(n)$. Нагадаємо, це є просто запис із модулярної арифметики [1]: $\kappa^r \equiv 1 + \cdot$, де κ - деяке ціле число. З іншого боку, з малої теореми Ферма випливає, що при цьому $r-1=n$ є просте число. Алгоритм пізніше було розширено, щоб знайти також дискретні логарифми в групах еліптичних кривих [4]. Було показано, що алгоритм дискретного логарифмування еліптичної кривої є більш ефективним, ніж алгоритм розкладання на множники, потенційно вимагаючи квантового комп’ютера з менш

ніж вдвічі меншою кількістю кубітів для великих значень n. Наприклад, розкладання 3072-бітного модуля RSA за оцінками вимагало приблизно 6144 логічних кубітів, у той час як для зламу настільки ж безпечного 256-бітного криптографічного ключа з еліптичною кривою потрібно лише 1800 логічних кубітів. Слід зазначити, якщо квантовий комп'ютер має певну кількість логічних кубітів, кількість фізичних повинна бути в кілька разів більшою. Таким чином, вже в найближчому майбутньому один із найпоширеніших криптографічних алгоритмів опиниться під загрозою злому.

Квантовий алгоритм, відомий як алгоритм Гровера [5], - це алгоритм пошуку, здатний знаходити елемент у невпорядкованій базі даних із $N = 2^n$ елементів лише за $O\left(\sqrt{N}\right)$ кроків на квантовому комп'ютері. Це квадратичне прискорення порівняно з класичним комп'ютерним підходом, який вимагає в середньому $N/2$ кроків із використанням лінійного пошуку. Цей квантовий алгоритм, реалізований на квантових комп'ютерах, можна використовувати для зламу криптосистем із симетричним ключем. Для захисту від атак на основі алгоритму Гровера нам потрібно подвоїти розмір ключа, щоб досягти подібного рівня безпеки, як проти звичайних комп'ютерів. Наприклад, для захисту від атак на основі квантового алгоритму Гровера систем із 128-бітним симетричним ключем нам потрібно використовувати криптосистеми із 256-бітним симетричним ключем. Також передбачається, що квантові комп'ютери зможуть зламати кілька сучасних криптографічних алгоритмів, які використовуються для захисту комунікацій через Інтернет, забезпечуючи необхідну довіру для безпечних транзакцій у цифровій економіці та для шифрування даних. Для захисту від атак квантових комп'ютерів постачальники продуктів безпеки та постачальники послуг повинні постійно оцінювати ризик, пов'язаний з вибором криптографічних алгоритмів, і розробляти абсолютно нові квантово-стійкі алгоритми для постквантового світу.

1.2 Постквантова криптографія

Постквантова криптографія (PQC) – це розробка криптографічних алгоритмів, які є безпечними в квантову еру з захистом як від класичних, так і від квантових комп’ютерів. Існує кілька підходів для побудови постквантових криптографічних схем. Це криптографія на основі хешу, криптографія на основі коду, багатоваріантна криптографія, криптогра́фія на ґра́тках та схеми криптографії на основі ізогенії.

1.2.1 Криптографія на основі хешу

Криптографія на основі хешування зосереджена на розробці схем цифрового підпису на основі безпеки криптографічних хеш-функцій, наприклад, SHA-3. Ці схеми ґрунтуються на звичайних вимогах до безпеки хеш-функцій і вимагають менших передумов про безпеку, ніж теоретико-числові схеми, наприклад, RSA або алгоритм цифрового підпису DSA. Ральф Меркл у 1989 році представив власну схему підпису MSS [6], яка заснована на одноразових підписах (наприклад, як схема підпису Лампорта [7]) і використовує бінарне хеш-дерево (дерево Меркла). Вважається, що MSS стійкий до квантових комп'ютерних алгоритмів [8].

У 2016 році Національний інститут стандартів і технологій (NIST) у Сполучених Штатах ініціював проект з оцінки та стандартизації одного або кількох постквантових алгоритмів відкритого ключа. Прийом робіт тривав до 2018 року. Після першого туру оцінювання 69 кандидатів продовжили участь у конкурсі, а після другого туру залишилося 26 кандидатів. Планується, що NIST представить перший постквантовий стандартний проект до 2025 року. Так підпис на основі хешу SPHINCS+ [9] обрано як альтернативне рішення в результаті третього раунду конкурсу NIST.

1.2.2 Криптографія на основі коду

Криптографія на коді має свою безпеку, спираючись на складність проблем із теорії кодування. Ця криптосистема заснована на кодах з виправленням помилок для побудови односторонньої функції [10]. Безпека базується на складності декодування повідомлення, яке містить випадкові помилки, і відновлення структури коду. Приватний ключ - це випадковий двійковий незвідний код, а відкритий ключ — матриця генератора випадково переставленої версії цього коду. Зашифрований текст - це кодове слово, до якого додано деякі помилки, і лише власник закритого ключа (код Goppa) може видалити ці помилки. Це старий підхід, тому через три десятиліття знадобилося деяке коригування параметрів, але відомо, що жодна атака не представляє серйозної загрози для цієї системи, навіть з боку квантового комп'ютера. Класична схема шифрування на основі цього коду була вибрана як схема-фіналіст у результаті третього раунду процесу стандартизації NIST.

1.2.3 Багатоваріантна криптографія

Безпека багатоваріантної (багатовимірної) криптографії залежить від складності вирішення багатовимірних систем рівнянь. Ці схеми базуються на системах багатовимірних поліноміальних рівнянь над скінченним полем. Існує кілька варіантів схем багатовимірної криптографії, заснованих на прихованих рівняннях поля (HFE), таких як, наприклад, незбалансовані криптосистеми «олії та оцту» (UOV). Система UOV використовується для підписів. Іншим прикладом багатоваріантної криптографії є Rainbow. Зокрема, Rainbow — один із фіналістів NIST. Детальніше про поточний стан схем багатоваріантної криптографії можна дізнатися з статті [11].

1.2.4 Криптографія на ґратках

Граткова криптографія є одним із найактивніших напрямків за останні роки з кількох ключових причин. По-перше, вона має сильні гарантії безпеки від деяких добре відомих проблем решітки, наприклад, проблеми найкоротшого вектора і проблеми кільцевого навчання з помилками. По-друге, це дозволяє застосувати потужні криптографічні примітиви, наприклад, функціональне шифрування. По-третє, деякі нові криптографічні схеми на основі решітки (гратки) стали досить практичними останнім часом, наприклад, протокол обміну ключами NewHope і схема підпису BLISS [12].

Криптографія на основі решітки є однією з успішних схем третього раунду процесу стандартизації NIST. В якості схем фіналістів обрано схеми шифрування на основі решітки Kyber, схеми підпису Dilithium [13], [28] і Falcon [14], [27].

1.2.5 Криптографія на основі ізогенії

Криптографія на основі ізогенії — це особливий тип постквантової криптографії, який використовує певні відповідні карти над кінцевими полями (як правило, еліптичними кривими) як основний будівельний блок. Його головними перевагами є відносно невеликі ключі та багата математична структура, які ставлять перед криптографами надзвичайно цікаві питання. Ці схеми засновано на ізогеніях суперсингулярної еліптичної кривої [15], які захищені від квантових супротивників. Ці схеми забезпечується в рамках задачі побудови ізогенії між двома суперсингулярними кривими з однаковою кількістю точок. Схеми, засновані на ізогенії, можуть служити цифровими підписами або обміном ключами, такі як схема суперсингулярної ізогенії Діффі-Хеллмана. Треба відмітити, що різновид цієї схеми SIKE [16] є єдиною схемою шифрування на основі ізогенії в альтернативному списку результатів третього раунду NIST. У результатах же третього раунду NIST не існує схеми підпису на основі ізогенії.

Таким чином, існує багато схем постквантвої криптографії. Треба сказати, що здебільшого вони є розвитком старих криптографічних схем. Практично всі алгоритми можна використовувати для цифрових підписів у тому чи іншому варіанті. Найчастіше вони й створювалися тільки з цією метою. У наступному розділі перейдемо до методів реалізації алгоритмів підписів.

1.3 Цифрові підписи

Цифрові підписи забезпечують спосіб забезпечення автентичності та цілісності повідомлень. Отримавши повідомлення (наприклад, лист, заяву або контракт), підписувач використовує свій секретний ключ, щоб створити підпис для його з’єднання. Будь-хто, хто володіє відповідним відкритим ключем підписанта, зможе перевірити походження та правильність повідомлення. Це доповнює секретність повідомлень, яка досягається шляхом встановлення спільного ключа через обмін ключами, а потім шифрування повідомлень за допомогою цього встановленого ключа. Разом ці фундаментальні примітиви складають основу практично всього безпечного спілкування в Інтернеті сьогодні.

1.3.1 Цифровий підпис на практиці

У цьому розділі ми розглядаємо використання цифрових підписів 14 реальними програмами в 4 широких категоріях (див. рисунок 1.2) [18]. Вибрані категорії: фінанси (для економіки), критична інфраструктура (для уряду, людей і пристроїв), Інтернет (для взаємодії «бізнес-бізнес», «бізнес-споживач», «рівноправний» і «Інтернет речей»).), і підприємство (для підприємств).

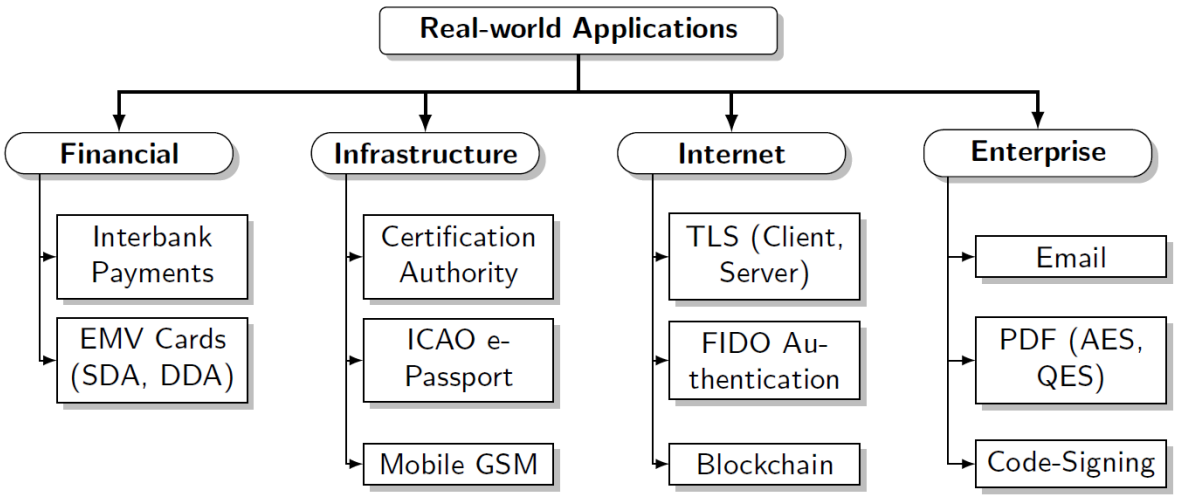

Рисунок 1.2 – 14 додатків, що використовують цифрові підписи [18](рисунок взято з інтернет порталу)

Хоча це не повне охоплення всіх випадків використання асиметричної криптографії, воно включає всі випадки використання цифрового підпису, знайдені у звіті Європейського інституту телекомунікаційних стандартів, і широкий пошук варіантів використання, який було проведено у пошуковій системі Google. У цьому дослідженні не розглядаються лише військові програми, оскільки такі вимоги зазвичай є засекреченими та мало публічних деталей. Таким чином, цифровий підпис використовується практично у всіх випадках, що забезпечують функціонування людської цивілізації. Іншими словами, якщо буде знайдено алгоритм взлома однієї схеми цифрової підписки, утрата довіри до безпеки приведе до катастрофічного порушення всієї системи безпеки. Основні загрози пов'язано з квантовими алгоритмами. Тому такий інтерес викликає розвиток постквантової криптографії.

1.3.2 Схема цифрового підпису

Схема цифрового підпису — це кортеж алгоритмів (KeyGen; Sign; Verify), визначених таким чином:

- Алгоритм генерації ключів KeyGen — це ймовірнісний алгоритм, який виводить відкритий ключ pk і секретний ключ sk, тобто пару ключів (pk; sk).

- Алгоритм підпису Sign — це можливо ймовірнісний алгоритм, який приймає як вхідні дані повідомлення і секретний ключ sk для створення підпису.

- Алгоритм перевірки Verify — це детермінований алгоритм, який приймає як вхідні дані повідомлення, підпис і відкритий ключ pk. Він виводить логічне значення True, щоб вказати, що підпис прийнято, або False, щоб вказати відхилення.

Для коректності ми вимагаємо, щоб для всіх (pk; sk) $\leftarrow$ KeyGen(), усіх повідомлень m і всіх підписів $\sigma \leftarrow$ Sign(m; sk) виконувалося Verify(m; σ ; pk) = True. Це означає, що всі правильно згенеровані підписи дійсно приймаються. Стандартним поняттям безпеки для схем підпису є екзистенційна неможливість підробки при атаках з адаптивним вибором повідомлень [17]. Інтуїтивно це поняття відображає ідею про те, що зловмисник не повинен мати можливість створювати дійсні підписи для будь-якого повідомлення m, навіть після отримання підписів на багатьох інших повідомленнях.

На рисунку 1.3 показано схему цифрового підпису з хешуванням повідомлення. Практично всі схеми підпису мають таку структурну схему, тому ми і навели зображення, як приклад, на основі хеш.

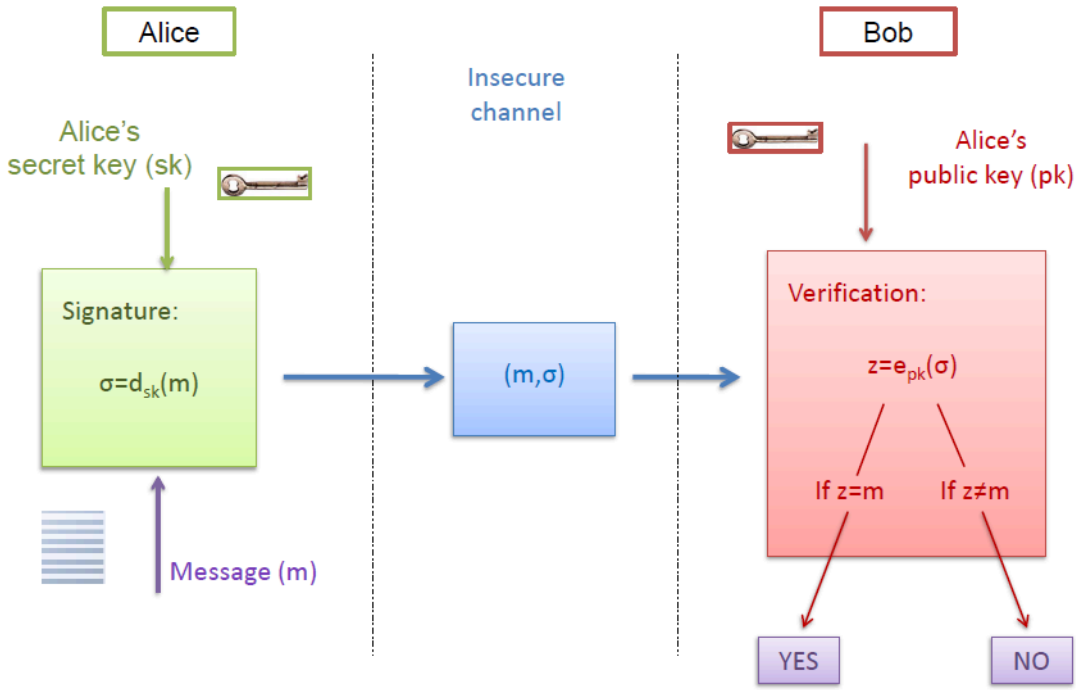

Рисунок 1.3 – Схема цифрового підпису (рисунок створено самостійно)

У розділі 1.2 були розглянуті різноманітні алгоритми постквантвої криптографії, тепер необхідно виділити ті, які виглядають найкращими для цифрового підпису.

1.3.3 Порівняльний аналіз алгоритмів цифрового підпису

Для розгляду ми виділили лише ті алгоритми, які переважно використовуються для цифрового підпису. Звичайно, вибір визначається поставленим завданням. Для цього послідовно розглянемо переваги та недоліки алгоритмів підпису з наступних розділів: криптографія на ґра́тках, багатоваріантна криптографія, на основі коду та криптографія на основі хешу.

Почнемо з криптографії на ґра́тках. Falcon [14], [27] і CRYSTALS-Dilithium [13], [28] — це два різні криптографічні алгоритми, запропоновані для

постквантової криптографії (PQC), галузі, призначеної для розробки криптографічних методів, стійких до атак квантового комп’ютера.

Falcon служить схемою цифрового підпису, яку засновано на складності проблеми навчання з помилками (LWE). Він розроблений для запобігання атакам як з класичних, так і з квантових комп’ютерів, з метою надання ефективних та безпечних цифрових підписів, які застосовано в реальних сценаріях, навіть за наявності квантових комп’ютерів .

CRYSTALS-Dilithium охоплює набір криптографічних алгоритмів, що базуються на проблемах решітки. Цей набір містить схеми шифрування, протоколи обміну ключами, схеми цифрового підпису та хеш-функції. Мета розробки CRYSTAL полягає в створенні криптографічних примітивів, несприятливих до квантових атак, що робить його придатним для розгортання в постквантовому криптографічному ландшафті.

Слід зазначити, що і Falcon, і CRYSTALS-Dilithium є невід’ємними компонентами наполегливих спроб криптографічного співтовариства розробити та стандартизувати нові алгоритми, здатні протистояти атакам з боку квантових комп’ютерів. Такі досягнення мають вирішальне значення, враховуючи потенційну вразливість чисельних зараз існуючих криптографічних схем до квантових атак.

До недоліків можна віднести розмір відкритого і закритого ключа. І ці схеми погано адаптовані для створення цифрового підпису. До того ж, алгоритми мають високу складність, та існує ймовірність виникнення помилки дешифрування правильно створеного зашифрованого тексту. Але є значні переваги:

- Ефективніше шифрування та дешифрування як у апаратній, так і в програмній реалізації.
- Набагато швидша генерація ключів, що дозволяє використовувати одноразові ключі.
- Низьке використання пам'яті дозволяє використовувати його в таких програмах, як мобільні пристрої та смарт-карти.

Слід визнати, що схема підпису BLISS [13] (теж на ґратках) виглядає щодо цього досить перспективною. Зокрема, вона має закритий ключ 7 кБ, відкритий 2 кБ та розмір підпису 5 кБ. Але знову ж таки її складність гальмує використання у багатьох областях.

Як другий кандидат, ми обираємо представника з табору багато інваріантної криптографії – це Rainbow [11]. Цей алгоритм використовується лише для підписів, у цьому його перевага. При цьому розмір відкритого ключа дорівнює 124 кБ, закритого - 95 кБ, і розмір підпису досить великий - 424 кБ. Хоча в його основі лежить складність розв'язання систем багатовимірних рівнянь, цей алгоритм страждає від ризику багатьох нападів, таких як: пряма атака, атака мінімального рангу, атака високого рангу, атака UOV, атака узгодження UOV, атаки на хеш-функцію.

Тепер ми розглядаємо переваги та недоліки цифрового підпису постквантової криптографії на основі коду. Перша криптосистема з відкритим ключем на основі коду була запропонована Робертом МакЕлісом ще у 1978 році [10] (дивиться розділ 2.1.2). Ця всеїдна криптографічна схема, вона успішно використовується як для шифрування, розшифровки та підпису. Припущення про надійність цієї криптосистеми полягає в тому, що декодування відомих лінійних кодів легко виконується ефективним алгоритмом декодування, але коли лінійний код маскується як загальний лінійний код за допомогою кількох секретних перетворень, декодування стає дуже складним. Щоб досягти визнання та уваги на практиці, постквантові схеми відкритого ключа мають бути ефективні. Крім того, їх реалізації мають працювати швидко, зберігаючи невеликі вимоги до пам’яті. Шифрування та дешифрування МакЕліса не потребує обчислювально дорогих арифметичних засобів множинної точності. Отже, він призначений також для реалізації на вбудованих пристроях.

Основним недоліком криптосистеми МакЕліса з відкритим ключем є дуже великий відкритий ключ. Наприклад, для 80-бітного рівня безпеки відкритий ключ, який використовується в оригінальній схемі, має розмір 437,75 кБ. Розмір

відкритого ключа другого варіанту становить, наприклад, 2,5 кБ, що в 175 разів менше. Тим не менш, одна з криптосистем цього сімейства успішно пройшла до третього раунду конкурсу NIST.

Тепер розглянемо схеми підпису на основі хешу, яка покладається виключно на існування безпечної односторонньої функції. Цей підхід часто вважається найбільш консервативним та доступним варіантом - не лише проти супротивника, оснащеного квантовим комп'ютером, але й порівняно з його класичними аналогами. Насправді було показано, що існування захищеної односторонньої функції є доведеною мінімальною вимогою для існування будь-якої схеми підпису, а для схем підпису на основі хешу це також достатньо. Підписи на основі хешування є одними з найстаріших криптосистем із відкритим ключем, і їхня безпека добре вивчена. На цьому етапі може виникнути природне запитання, чому ці схеми ще не були широко розгорнуті для надання цифрових підписів у присутності класичних противників. Причин цьому, на жаль, багато. Історично великий розмір підпису був важливою перешкодою для практичних схем. Але можливо, найбільша перешкода, однак, мала зовсім іншу природу: виробники хеш-підписів повинні були підтримувати та оновлювати стан цієї системи. Замість того, щоб мати можливість створити пару ключів один раз, а потім використовувати їх довільно та нескінченно, секретний ключ ефективно змінюється з кожним підписом. Ця, здавалося б, нешкідлива різниця має серйозний вплив на реальні програми та прямо суперечить типовим очікуванням користувачів.

Тем не менш, багато проблем цього сімейства алгоритмів були вирішені останнім часом. Так система SPHINCS+ [9] володіє дуже вражаючими характеристиками: розмір відкритого ключа 32 Б, закритого 64 Б і розмір підпису всього 8 кБ. Але є і інші переваги:

- Найкраща альтернатива підпису з теорії чисел.
- Підписи використовуються малого та середнього розміру.
- Невеликий розмір ключа.

Постійно зростаюча загроза великомасштабного квантового комп’ютера відновила інтерес до цієї галузі, і за останнє десятиліття відбувся значний розвиток. Це призвело до XMSS [19], що демонструє життєздатність підписів на основі хешування, а згодом і до SPHINCS [9], [26], в якому не потрібно підтримувати стан системи.

Виходячи з цього, у цій роботі ми досліджуємо алгоритм цифрового підпису на основі хеша у із сукупності алгоритмів PQC.

1.4 Постановка задачі

У ході даної кваліфікаційної роботи потрібно провести дослідження предметної області застосування постквантових алгоритмів цифрового підпису, аналіз методів і засобів створення сучасних систем підпису, які стійкі до квантових алгоритмів. Буде проведено порівняльний аналіз різних постквантових алгоритмів цифрових підписів, порівняння їх з огляду на ефективність, продуктивність та стійкість до квантових атак. Ця робота спрямована на виявлення найкращих алгоритмів для створення сучасних систем цифрового підпису, стійких до квантових обчислень, та буде корисною для криптографів і розробників систем безпеки.

Реалізація наукового дослідження складається з наступних етапів: Аналіз предметної області – методів побудови постквантових систем підпису.

- Дослідження архітектурних особливостей сучасних схем цифрового підпису розгляд їх основних особливостей.
- Порівняльний аналіз сучасних постквантових алгоритмів цифрового підпису та виведення підсумків за вибором.
- Моделювання та порівняльний аналіз постквантових алгоритмів.
- Практична реалізація – розробка моделей обраних схем цифрового підпису.
- Проведення експерименту – оцінка ефективності реалізованих моделей.

2 АЛГОРИТМИ ПІДПИСІВ НА ОСНОВІ ХЕШУ

У постквантовій криптографії (PQC) підпис на основі хешу (HBS, hash-based signature) є перспективним кандидатом для забезпечення безпечних цифрових підписів. Вони покладаються на властивості односторонніх хеш-функцій, які вважаються безпечними як для класичних, так квантових комп'ютерів. Підписи на основі хешу привабливі ще й тому, що вони прості, швидкі та ефективні. Вони можуть бути реалізовані з відносно невеликими розмірами ключів та вимагають мінімальних обчислень як для підпису, так і для перевірки.

Вивчення хеш-функцій вже є ключовим аспектом криптографії [1], і існують різні методи та дослідження, спрямовані на досягнення різних властивостей безпеки. На відміну від цифрових підписів, заснованих на складних задачах теорії чисел, якщо хеш-функцію атакують, це не порушить загальну безпеку HBS. Атаковану хеш-функцію можна легко замінити безпечною. Хеш-функції були вдосконалені та ефективно реалізовані протягом кількох десятиліть, що робить підписи на основі хешів дуже ефективними. Крім того, вибираючи різні основні хеш-функції та їхні параметри, можна збалансувати розмір підпису, час і пам’ять, щоб задовольнити різноманітні потреби програм. Зокрема, враховуючи, що симетричні ключові алгоритми вважаються безпечними проти квантового супротивника (через фантастичні вимоги до ресурсів), інтуїтивно зрозуміло використовувати ці алгоритми для розробки асиметричних ключових альтернатив.

Ми вже згадували у розділі 1.3.3, що основною перешкодою для використання HBS є необхідність постійної підтримки працездатності системи підписів на основі хеша. Тем не менше, протягом всієї історії розвитку криптографічної схеми підписів інтерес до ніколи не угасал. Хронологія для підписів на основі хешу наведена на рисунку 2.1 [20].

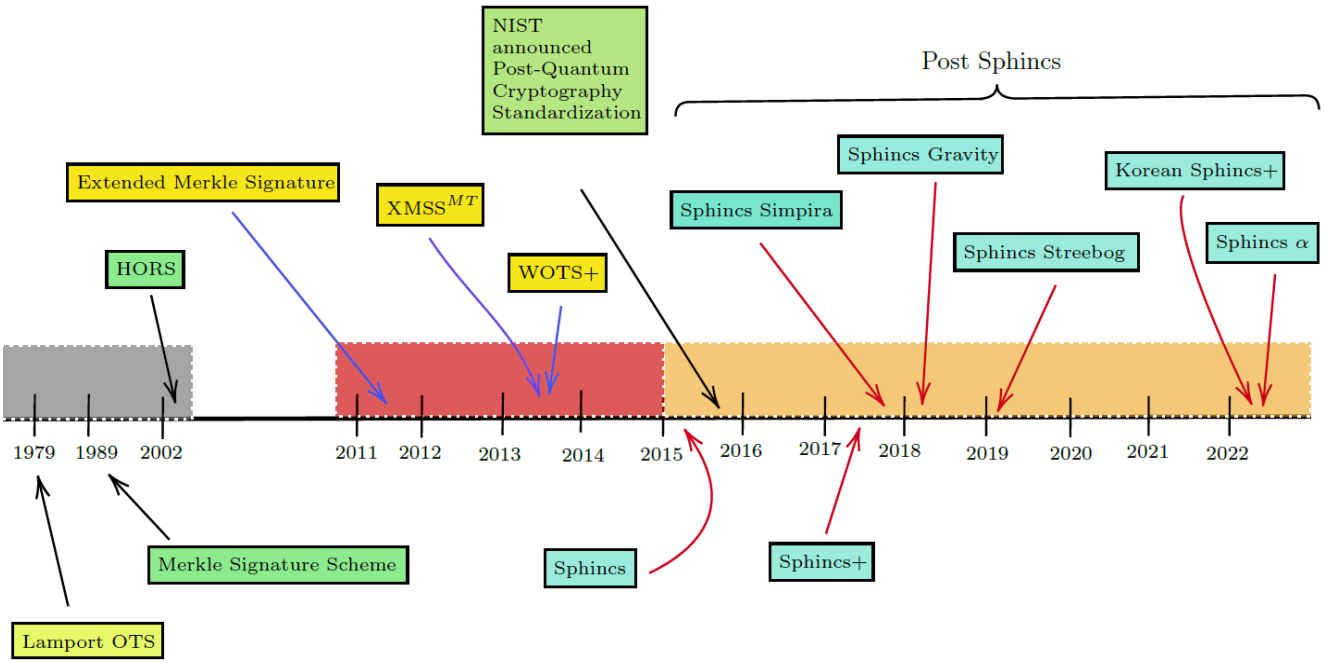

Рисунок 2.1 – Хронологія HBS (рисунок створено самостійно)

Як видно, в останні роки основна увага була прикута до алгоритмів із сімейства SPHINCS [9], що співпадає з думкою експертів NIST, але не можна забувати про оригінальні схеми на деревах Меркля XMSS [6,19] і про те, що основні підходи були сформовані в період розвитку системи одноразових підписів (OTS, one-time signature) Лампорта [7].

2.1 Схеми OTS

У розділі 1.3.2 (дивиться ще рисунок 1.2) було введено поняття цифрових підписів. Підписувач створює цифровий підпис на повідомленні за своїм вибором, використовуючи свій секретний ключ. Після цього верифікатор може використати відповідний відкритий ключ, щоб перевірити, чи підпис створено за допомогою цього конкретного секретного ключа та відповідає цьому конкретному повідомленню. Це відношення принципово асиметричне: лише власник секретного ключа може створити підпис, тоді як будь-хто, хто має доступ до

відкритого ключа, може перевірити правильність підпису для даного повідомлення.

Для схем OTS вводиться додатковий елемент асиметрії. Як випливає з назви, підписувач може використовувати свій секретний ключ лише один раз, тобто використовувати його лише для створення одного підпису. Безпека погіршується, коли одна пара ключів використовується для декількох повідомлень. Ситуація для верифікатора не змінилася: вони можуть вільно використовувати відкритий ключ для перевірки кількох повідомлень.

2.1.1 Схема OTS Лампорта

Відомо, що перша OTS на базі хешу була створена Лампортом [7], в подальшому вона була удосконалена Вінтерніцом[26] (WOTS, Winternitz one-time signature).

Припустимо, що маємо повідомлення $m \in \{0,1\}^n$ - тобто двійкове повідомлення m з n біт. Далі визначимо функцію $F : \{0,1\}^k \to \{0,1\}^k$, яка є односторонньою функцією, тобто, враховуючи значення $y = F(x)$, неможливо знайти інше x', щоб $F(x') = y$. Нагадаємо, що, наприклад, хеш-функція SHA-256 може обробляти різні вхідні дані та генерувати 256-бітові вихідні.

Перед тим, як отримати можливість створювати підписи, отправителю необхідно створити секретний ключ і відповідний відкритий ключ. Секретний ключ складається з 2n випадкових значень по k біт кожного. Ми маркуємо їх як $s_{i,j}$ для $i \in \{0,...,n-1\}$ і $j \in \{0,1\}$. Відкритий ключ складається з результатів застосування F до кожного секретного значення. Далі отримуємо $p_{i,j} \leftarrow F(s_{i,j})$ і публікуємо відкритий ключ $pk = (p_{0,0}, p_{0,1}, ..., p_{n-1,0}, p_{n-1,1})$.

Отримавши повідомлення m, підписувач вибірково відкриває секрети $s_{i,j}$, що відповідає значенням бітів в m. Точніше, це значення s_{i,m_i}, де m_i — i-й біт m. Список цих значень складає підпис. Таким чином, пусть у нас є чотирибітове повідомлення 1001. І ми маємо секретний ключ, який складається з восьми випадкових k-бітових рядків $s_{i,j}$, де $i \in \{0,...,3\}$, а $j \in \{0,1\}$. Пам'ятаємо, що при використанні хеш-функції SHA-256 к = 256. Тоді на першому місці в повідомленні стоїть 1, вибираємо рядок секретного ключа $s_{\text{місце,значення}} = s_{0,1}$, для другого - $s_{1,0}$, для третього , очевидно, $s_{2,0}$ та для четвертого $s_{3,1}$. Це і буде підписом до повідомлення 1001.

Структура підпису Лампорта показано рисунку 2.2. Публічний ключ залишився без зміни. А ось за допомогою свого закритого ключа ми зробили підпис. Список цих значень складає підпис, який надсилається одержувачу разом із повідомленням. Враховуючи такий підпис, верифікатор також може застосувати F до розкритого секрету та отримати $F(s_{i,m_i})$. Підпис слід вважати дійсним, якщо дійсно $F(s_{i,m_i}) = p_{i,m_i}$ для всіх бітів повідомлення m.

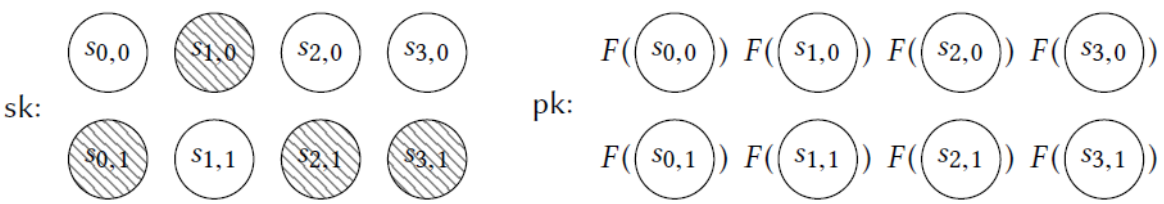

Рисунок 2.2 – Схема підпису Лампорта. Тут m = 1011 використовується як приклад повідомлення, а сірі вузли вказують на розкриті секрети (рисунок створено самостійно)

Як наслідок односпрямованості F, зовнішній фальсифікатор не може створити такий підпис для будь-кого іншого m: вони не знають жодних секретів

$s_{i,j}$, крім тих, що точно відповідають бітам m. Зміна хоча б одного біта в повідомленні m потребує включення нерозкритого секрету в підпис.

Інтуїтивно зрозуміло, чому такий секретний ключ не можна використовувати для декількох повідомлень. Якби ми також опублікували підпис для іншого повідомлення m' на основі тих же секретних значень $s_{i,j}$, сторонній спостерігач, знаючи вже секрети s_{i,m_i}, дізнався б також s_{i,m'_i}. Це не тільки дозволяє їм відтворити підпис для m і m' (що не є проблемою, оскільки вони вже були законно опубліковані), але також для будь-якого повідомлення m'', яке можна створити шляхом поєднання бітів з m і m'.

У зв’язку з тим, що в нашій схемі розмір ключа дорівнює розміру повідомлення, а в деяких протоколах і більше, розробка цифрової підписки документа великого розміру – досить складний і довгий процес. Тому зазвичай до тексту документа спочатку застосовують більш швидке і просте хешування. Отриманий порівняно короткий результат шифрують закритим ключем, одержуючи сам цифровой підпис, який разом з відкритим текстом документа передається отримувачу. Таким чином, на прикладі, який було розглянуто вище, ми генеруємо цифровий підпис не з усього документа, а лише на його хеш-образі. По каналу зв'язку одержувачу передається весь документ та його цифровий підпис. Одержувач здійснює хешування документа та перевіряє справжність підпису за відомим вже алгоритмом. Перевірити справжність цифрового підпису може будь-хто. Ніякого шифрування тут немає. Це просто протокол для автентифікації.

Крім безумовного недоліку, що можна підписати лише один раз на пару ключів, підписи Лампорта також досить великі. При 2kn бітах підпис 256-бітного хешу з k = 256 призводить до підписів 128 КіБ кожен. Схема [6] Меркла демонструє невелике покращення в порівнянні зі схемою Лемпорта. Замість того, щоб розкривати секрети як для 1-бітів, так і для 0-бітів, лише розкриття секретів, які відповідають 1-бітам, зменшує розмір підпису в середньому вдвічі. Щоб

запобігти підробкам, просто пропускають відповідний секрет із підпису, загальна кількість нулів у m додається до m і також підписується.

2.1.2 Схема Вінтерніца

Також у [6] Меркл описує варіант алгоритму підпису, приписуваний Вінтерніцу, який дозволяє зменшити розмір підпису за рахунок додаткового часу виконання. Це схема WOTS[26]. Розвиток полягає в тому, що замість використання секрету для автентифікації одного біта повідомлення можна охопити кілька бітів повідомлення, використовуючи один секрет і різну кількість застосувань односторонньої функції. Застосування односторонньої функції зазвичай називають ланцюжками, а саму функцію - ланцюговою функцією.

Як приклад, розглянемо випадок, коли $w = 2^4 = 16$, тобто випадок, коли біти згруповані в групи по чотири. Для повідомлення з n біт нам знадобиться лише n/4 секрети. Для кожної такої групи підписувач застосовує односторонню функцію F так часто, як вказує значення групи. Зауважте, що максимальне значення дорівнює w − 1. Отже публічні значення мають бути $p_i = F^{w-1}(s_i) = F^{15}(s_i)$. Щоб підписати групу mi = 1101, тобто десяткове значення 13, підписувач повинен обчислити та отримати підпис $\sigma_i = F^{13}(s_i)$. Це залишає w−1−13 = 2 застосування F, після чого отримувач (веріфікатор) може перевірити, що справді $p_i = F^2(\sigma_i) = F^2(F^{13}(s_i))$.

Розглянемо механізм WOTS[26] на найпростішому прикладі. Припустимо, що Аліса хоче підписати своє повідомлення Бобу, як на рисунку 1.2. Як завжди, процес можна пояснити у три кроки: генерація ключа, генерація підпису та перевірка підпису.

Генерація ключів:

- Алісі потрібно створити пару ключів – закритий і відкритий ключ.

- Для створення закритого ключа використовується генератор випадкових чисел для генерації 32 256-бітних випадкових чисел. Закритий ключ відомий лише Алісі.
- Щоб створити відкритий ключ, кожне з 32 чисел хешується 256 разів, щоб отримати інший набір із 32 256-бітних чисел. Відкритий ключ надається всім.

Генерація підпису:

- Аліса хешує повідомлення за допомогою SHA 256, який створює 256-бітний блок. Цей блок, дивиться рисунок 2.3, розбивається на 32 8-бітних значення (N1, N2, …, N32).
- Аліса хешує кожне 8-бітне значення 256-N разів, де N є значенням 8-бітного значення. Наприклад, якщо N1 є 8-бітним значенням 10001000 = 136, тоді N1 буде хешовано 256-136 = 120 разів. Після цього для кожного з 8-бітних значень буде згенеровано цифровий підпис

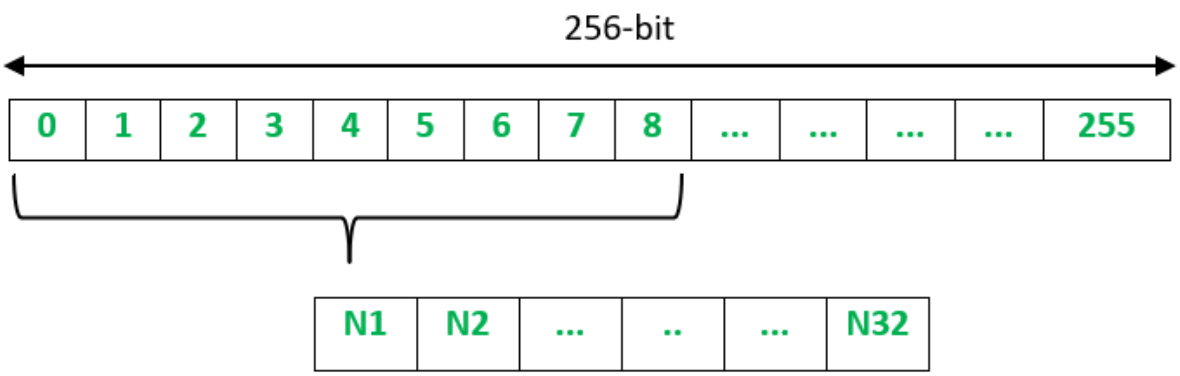

Рисунок 2.3 – Приклад розбиття хешу (рисунок створено самостійно)

Верифікація підпису:

- Боб хешує повідомлення за допомогою SHA-256, щоб створити блок (дайджест) з 32 8-бітних значень (N1, N2, …, N32).
- Потім Боб хешує значення підпису на кількість разів, вказану в хеш-значенні повідомлення (N1, N2, …, N32).

- Боб порівнює результат із відкритим ключем Аліси. Якщо вони збігаються, підпис дійсний.

Враховуючи підпис WOTS, можна повністю реконструювати відкритий ключ, що відповідає секретному ключу, який використовувався для створення підпису. Фактично, можливо помітити, що це саме те, що відбувається під час процедури перевірки . Під час перевірки, одностороння функція повторно застосовується до всіх включених значень. Це продовжується рівно до тих пір, поки вихід не збігається з відкритим ключем; фактична «перевірка» зводиться до порівняння отриманих значень із попередньо отриманою копією ключа, щоб підтвердити це. Зауважимо, що це можна зробити ще далі, обчисливши дайджест за відкритим ключем і порівнявши його з попередньо отриманим автентичним дайджестом. Це ефективно стискає ключ до розміру одного дайджесту.

У наступному розділі ми побачимо, як ця, здавалося б, нічим не примітна властивість надзвичайно корисна при побудові схеми багаторазового підпису з WOTS.

2.2 Схеми багаторазових підписів

Якщо припустити, що перевіряючий має доступ до необхідного відкритого ключа, одноразовий підпис забезпечує ту саму функціональність, яку ми очікуємо від будь-якої схеми цифрового підпису. Однак це припущення вражає хворе місце: оскільки пару ключів можна використати лише один раз, для кожної операції перевірки підпису потрібен інший відкритий ключ. Це легко виправити, додавши відповідний відкритий ключ до одноразового підпису, лише для виявлення справжньої проблеми: автентифікації вкладених відкритих ключів. Усі схеми підписів, які ми бачили досі, були схемами одноразового підпису. Як правило, це не той примітив, який корисний на практиці. Тепер ми обговоримо, як побудувати багаторазову схему підпису зі схеми одноразового підпису, як запропоновано Мерклем [6].

2.2.1 Дерева Меркла

Найбільша проблема схем OTS – керування ключами. Обмін відкритим ключем дуже складний. Повинне бути гарантовано, що відкритий ключ належить передбачуваному партнерові зв'язку і що відкритий ключ не було змінено. Тому слід використовувати невелику кількість відкритих ключів, а відкриті ключі мають бути досить короткими. Але у схемах OTS в кожному підпису використовується новий відкритий ключ, і це відкритий ключ досить великий проти інших схем підпису. Щоб зробити схеми OTS здійсненими, необхідно ефективне керування ключами, яке зменшує кількість відкритих ключів та їх розмір. У [6] Меркл представив схему підпису MSS, в якій один відкритий ключ використовується для підпису багатьох повідомлень. Основна ідея полягає в тому, щоб використовувати листя дерева для зберігання ключів OTS. Оскільки дерева Меркла є бінарними деревами, дерево Меркла висотою n має 2^n листя. Листя використовуються для керування дайджестами ключів OTS, а само дерево може керувати 2^n парами ключів OTS. Під час підписання повідомлення з дерева потрібно вибрати одну пару ключів OTS, яка раніше не використовувалася. Підпис складається з індексу аркуша, відкритого ключа OTS, дайджесту відкритого ключа OTS (листка) та шляху автентифікації цього аркуша.

Процес генерації ключів у схемі MSS відбувається в такий спосіб (дивиться рисунок 2.4). Як ми згадували вище, схема підпису Меркла може використовуватись лише для підпису обмеженої кількості повідомлень за допомогою однієї публікації відкритого ключа. Кількість можливих повідомлень має бути ступенем двох, тому ми позначаємо можливу кількість повідомлень як $N = 2^n$. Першим кроком створення відкритого ключа є створення відкритих ключів Y_i і закритих ключів X_i N одноразових підписів, як описано в розділах

2.1.1 і 2.1.2 (там ми використовували позначення p_i і s_i). Для кожного відкритого ключа Y_i з $1 \leq i \leq N$ обчислюється хеш-значення $h_i = H(Y_i)$. З цими хеш-значеннями будується дерево Меркля (також зване хеш-деревом). Ми називаємо вузол дерева $a_{i,j}$, де і позначає рівень вузла. Рівень вузла визначається відстанню від вузла до листка. Отже, листок дерева має рівень i = 0, а корінь – рівень i = n. Ми пронумеруємо всі вузли одного рівня зліва направо, щоб $a_{i,0}$ був крайнім лівим вузлом рівня і. На дереві Меркла хеш-значення h_i є листками бінарного дерева, так що $h_i = a_{0,i}$. Кожен внутрішній вузол дерева є хеш-значенням конкатенації двох своїх дочірніх елементів. Таким чином, $a_{1,0} = H(a_{0,0} \| a_{0,1})$ та $a_{2,0} = H(a_{1,0} \| a_{1,1})$, де || - це позначення конкатенації хешів (відрізняються, наприклад, між MD, SHA...). Приклад дерева Меркля з вісьмома листками зображено на рисунку 2.4.

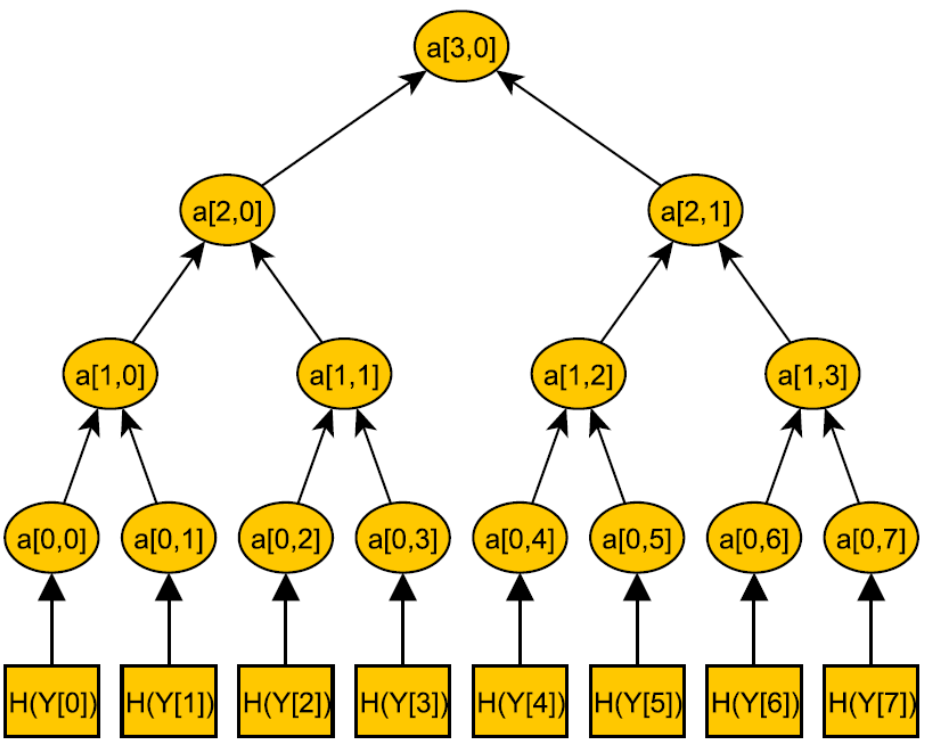

Рисунок 2.4 – Дерево Меркля з вісьмома листками (рисунок взято з інтернет журналу bitcoinmagazine)

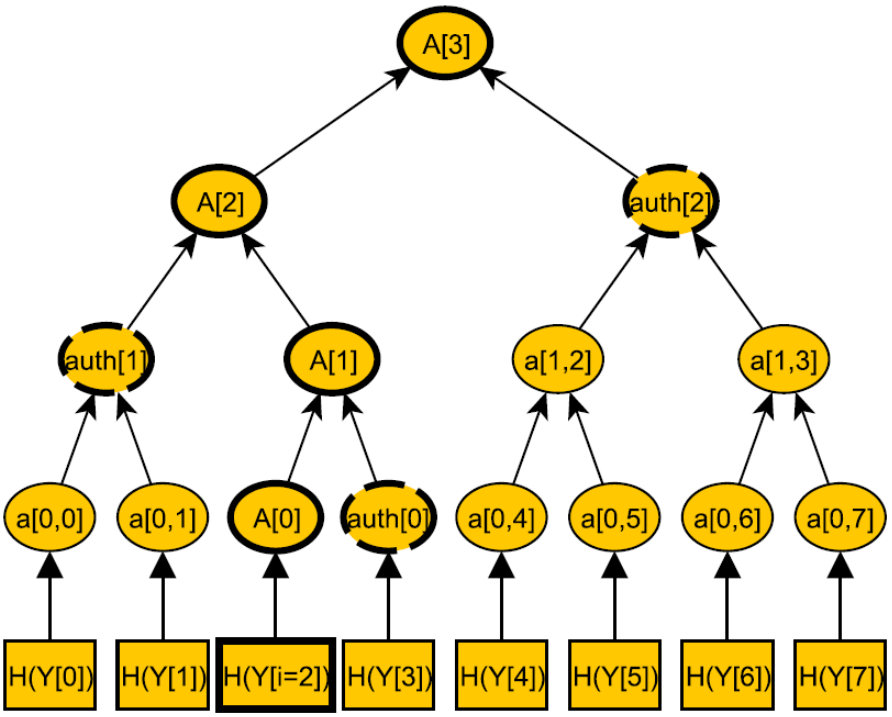

Рисунок 2.5 – Дерево Меркля із шляхом A та шляхом автентифікації для i=2 (рисунок взято з інтернет журналу bitcoinmagazine)

Щоб підписати повідомлення M за допомогою схеми MSS, повідомлення M підписується одноразовою схемою підпису, що призводить до підпису σ'. Це робиться за допомогою однієї з відкритих і закритих пар ключів (X_i, Y_i). Відповідний лист хеш-дерева для одноразового відкритого ключа Y_i є $a_{0,i} = H(Y_i)$. Ми називаємо шлях у хеш-дереві від $a_{0,i}$ до кореня як A. Схема дерева Меркля із шляхом A та шляхом автентифікації для i=2 зображена на рисунку 2.4. Шлях A складається з n + 1 вузлів, $A_0,.....A_n$, де $A_0 = a_{0,i}$ є листом, а $A_n = a_{n,0}$ є коренем дерева (публічним ключем). Щоб обчислити цей шлях A, нам потрібні всі дочірні елементи вузлів $A_0,.....A_n$. Ми знаємо, що A_i є нащадком A_{i+1}. Щоб обчислити наступний вузол A_{i+1} шляху A, нам потрібно знати обох нащадків A_{i+1}. Отже, для цього потрібен братський вузол A_i. Ми

називаємо цей вузол auth_i, так що $A_{i+1} = (A_i \| \text{auth}_i)$. Отже, n вузлів $\text{auth}_0,...\text{auth}_{n-1}$ необхідні для обчислення кожного вузла шляху A. Тепер ми обчислюємо та зберігаємо ці вузли $\text{auth}_0,...\text{auth}_{n-1}$. Як це зробити ефективно, це окреме завдання. Ці вузли, а також одноразовий підпис σ' повідомлення M формують остаточний підпис $\sigma = (\sigma' \| \text{auth}_2 \| \text{auth}_3 \| ... \| \text{auth}_{n-1})$ з схеми MSS.

Далі одержувачу необхідно верифікувати підпис. Одержувач отримує pub відкритий ключ, повідомлення M та підпис $\sigma = (\sigma' \| \text{auth}_2 \| \text{auth}_3 \| ... \| \text{auth}_{n-1})$. Спочатку одержувач перевіряє одноразовий підпис σ' повідомлення M. Якщо σ' є дійсним підписом M, одержувач обчислює $A_0 = H(Y_i)$ шляхом хешування відкритого ключа одноразового підпису. Для j =1, .., n−1, вузли A_j A обчислюються за $A_j = H(a_{j-1} \| b_{j-1})$. Якщо A_n дорівнює відкритому ключу схеми MSS, підпис дійсний.

Великою перевагою схеми MSS є те, що багато підписів можна створити за допомогою лише одного відкритого ключа. Однак ця перевага пов’язана зі збільшенням часу обчислення та довжини підпису. Далі ми розглянемо час обчислення кожної частини процесу підпису. Щоб зробити відкритий ключ, потрібно згенерувати 2^n одноразових ключів підпису. Потім необхідно обчислити кожен вузол хеш-дерева. Дерево складається з $2^{n+1} - 1$ вузлів. Для обчислення вузла потрібна одна хеш-операція, тому для генерації відкритого ключа потрібно $2^{n+1} - 1$ хеш-операція. Очевидно, що розміри такого дерева обмежені. Обчислити 2^{40} вузлів дуже дорого, обчислити 2^{80} вузлів неможливо.

Для створення підпису потрібні вузли $\text{auth}_0,...\text{auth}_{n-1}$. Якщо ви не зберігаєте вузли дерева, вузли потрібно генерувати знову для кожного підпису. Створення дерева дуже дороге, тому генерація всього дерева для кожного підпису

є непрактичною для великих дерев. Збереження всіх $2^{n+1}-1$ вузлів призведе до великих потреб у пам'яті. Отже, необхідна хороша стратегія для створення підпису, щоб не зберігати надто багато вузлів, але достатньо, щоб все проходило у все ще ефективний час. Ця проблема називається проблемою обходу **(**traversal problem**)** дерева Меркла.

Час перевірки досить швидкий порівняно з часом підпису. Спочатку необхідно перевірити одноразовий підпис. Після цього слід обчислити шлях $A = A_1,.....A_n$. Для цього потрібно лише n хеш-операцій, по одній для кожного вузла.

Підпис схеми MSS складається з одноразового підпису σ' і n вузлів $auth_0,...auth_{n-1}$. Якщо використовується 256-бітна хеш-функція, розмір підпису буде $|\sigma|=|\sigma'|+256*n$.

2.2.2 Схема підпису SPHINGS

SPHINCS[26] - це хеш-схема підпису, заснована на підході дерева Меркла. Як і MSS, безпека SPHINCS покладається виключно на властивості криптографічної хеш-функції. Хеш-дерево SPHINCS є гіпердеревом, що складається з кількох шарів хеш-дерев. Листові вузли верхніх шарів у гіпердереві містять ключі WOTS, які використовуються для підпису кореневих вузлів дерев на нижчому рівні. Схема багаторазового підпису на базі хешу для отримання випадкової підмножини (HORS, Hash to Obtain Random Subset) з деревами HORS (HORST, HORS tree) використовується для підпису повідомлень, а ключі HORST містяться в листових вузлах найнижчого шару дерев.

SPHINCS, порівняно з XMSS та багатьма іншими схемами підпису на основі хешу, є повністю схемою цифрового підпису без збереження стану. Інші схеми підпису на основі хешу, засновані на деревах Меркля, зберігають лічильник

листового індексу як частину закритого ключа. Цей лічильник оновлено для кожного нового згенерованого підпису, щоб запобігти повторному використанню тієї самої пари ключів. Незалежність від попереднього стану в схемі SPHINCS досягається випадковим вибором аркушевого індексу, що відповідає парі ключів, незалежно від того, чи використовувалася ця пара ключів раніше. Цей рандомізований вибір листкового індексу, природно, створює ризик повторного використання тієї самої пари ключів кілька разів. Якщо використовувати схему одноразового підпису, це призведе до повної компрометації системи. Однак, оскільки SPHINCS використовує схему багаторазового підпису, ця загроза зведена до мінімуму.

Схема малоразового підпису, така як HORST, подібна до схеми OTS, але її можна використовувати для підпису кількох повідомлень без шкоди для закритого ключа. Проте з кожним підписаним повідомленням зростає ймовірність можливої підробки. Параметри для SPHINCS-256 вибрано таким чином, що ймовірність підробки завжди повинна бути достатньо малою для забезпечення 128-бітної безпеки від квантових зловмисників. Доведено, що навіть якщо підписано 2^{50} повідомлень, що займе більше 30 років, якщо підписувати мільйон повідомлень в секунду, ймовірність успішної постквантової атаки має бути нижче 2^{-48}. Однак безпека погіршується зі збільшенням кількості підписів, що, отже, означає, що кількість підписів які можна створити обмежені.

На першому етапі в SPHINCS генерується пара секретних ключів $(\mathrm{sk}_1, \mathrm{sk}_2) \in \{0,1\}^n \times \{0,1\}^n$. sk_1 використовується для генерації псевдовипадкового ключа, а sk_2 використовується для генерації непередбачуваного індексу та рандомізації хешу. Крім того, генерується невелика кількість бітових масок Q. Бітові маски використовуються для всіх екземплярів WOTS і HORST, а також для дерев.

Щоб згенерувати відкритий ключ, необхідно згенерувати кореневий вузол. Для цього генеруються пари ключів WOTS для самого верхнього дерева. Усі ключі

генеруються псевдовипадковим чином із sk_1. Листки, що складаються з відкритих ключів WOTS, використовуються для побудови бінарного хеш-дерева та обчислення значення кореневого вузла pk_1. Приватний ключ складається з (sk_1 ; sk_2 ; Q), а відкритий ключ складається з (pk_1 ; Q).

Підпис на повідомлення $m \in \{0,1\}^*$ генеруються шляхом спочатку генерації псевдовипадкового значення $R = (R_1, R_2)$ шляхом подачі m і sk_2 у псевдовипадкову функцію. Рандомізований дайджест повідомлення D обчислюється як рандомізований хеш m, використовуючи R_1 як випадковість. Щоб підписати дайджест повідомлення D, необхідно вибрати пару ключів HORST. Індекс, який використовується для вибору пари ключів HORST, обчислюється за допомогою R_2. Індекс вибирає як дерево, так і індекс листа всередині вибраного дерева.

Підпис SPHINCS містить індекс, випадковість R_1 і підпис HORST. Крім того, для перевірки підпису потрібен один підпис WOTS+ і шлях автентифікації на кожен шар дерев. Ці значення обчислюються під час процесу підписання шляхом генерації одного бінарного хеш-дерева для кожного шару гіпердерева SPHINCS. Підписання є детермінованим, оскільки вся необхідна випадковість генерується за допомогою псевдовипадкової функції.

Щоб перевірити підпис SPHINCS, верифікатор повинен перевірити підпис HORST і один підпис WOTS+ і шлях автентифікації на кожен шар дерев. Роблячи це, верифікатор може обчислити значення для кореневого вузла. Підпис дійсний, якщо обчислене значення дорівнює значенню PK1 відкритого ключа.

Схеми підпису на основі хешу мають дві властивості, які не стосуються інших схем підпису. Перший полягає в тому, що схеми підписів на основі хешу мають обмежену кількість підписів, які можна створити. По-друге, XMSS є схемами зі збереженням стану, тобто закритий ключ оновлюється після кожного створеного підпису.

Для XMSS максимальна кількість підписів, які можна створити, визначається під час генерації ключа. Використовуючи основні набори параметрів, можна створити десь від 2^{10} підписів за допомогою XMSS. Вибір більшого числа негативно впливає на час роботи та розмір підпису. Для SPHINCS-256 обмеження підпису спричинено погіршенням безпеки для кожного створеного підпису. Не рекомендується використовувати ту саму пару ключів для більш ніж 2^{50} підписів, оскільки в цьому випадку безпека значно погіршується.

XMSS є схемами з підтримкою стану, що робить їх непридатними для систем, які потребують резервного копіювання або для яких слід розповсюджувати підпис. SPHINCS немає цієї проблеми, але оскільки безпека погіршується зі збільшенням кількості згенерованих підписів, розподілене підписання все ще може бути проблематичним.

Таким чином у цьому розділі були розглянуті основні схеми підписки на основі хеша. Якщо ми розглядаємо схеми OTS, практично всі моделі пов'язані зі схемою Лампорта. Вона досить ефективна, широко використовується. Модифікація Вінтерниці не порушує основні концепції підходів одноразових підписів, а просто використовує механізм економії ресурсів. Тем не менш, на основі цієї модифікації були розроблені схеми багаторазових підписів, засновані на концепції дерева Меркля, в яких широко використовуються схеми OTS. Дерево Меркля лежить в основі більш продвинутих (і більш складних) схем підписів. Це перш за все сім'я SPHINCS. Резюмуючи, можна відзначити, що основні проблеми сучасних схем підписів такі ж, як і проблеми, які зустрічаються при розгляді дерева Меркля. Це перш за все проблема аутентифікації шляхів і проблема обходу.

3 ІМПЛЕМЕНТАЦІЯ АЛГОРИТМІВ HBS

Спочатку ми познайомимося з алгоритмом хешування дерева, щоб ефективно обчислити вузол у дереві хешування. Цей алгоритм буде використано пізніше для створення відкритого ключа та створення наступного шляху автентифікації. Але перед цим познакомимся з кодами по створенню від Лампорта, він є фундаментом для побудови дерева Меркла. Далі розглянемо код побудови самого дерева Меркла.

3.1 OTS Лампорта

Схема OTS Лампорта створює 256 пар 256-бітних закритих ключів (у парах у позиціях 0 і 1), а односторонній хеш обчислюється для створення масиву з наступних 256 пар 256-бітних відкритих ключів. Весь масив відкритих ключів 512x256 біт є відкритим ключем. Підпис передбачає створення 256-бітного хешу повідомлення, а потім побітову перевірку результату хешу для кожного біта - якщо 0, підписом є перший закритий ключ, якщо 1, як підпис вибирається другий. Очевидно, що це одноразовий підпис. Розмір відкритого ключа можна зменшити до 256 біт, включивши 2,4,8 тощо підписів OTS у дерево Меркла.

Отже, необхідно зробити такі кроки:

- Потрібно створити ключ підпису.
- Використати ключ підпису та створити ключ перевірки.
- Беремо повідомлення та використовуємо ключ підпису для створення підпису.
- Отримуємо повідомлення та підпис і використовуємо ключ перевірки для перевірки підпису.

Експериментувати краще, використовуючи Python, а самі коди будувати на основі інтерпретованої мови програмування. Зазвичай для цього вибирається C.

Насправді Python багато бібліотеки написані на основі C. За основу беремо код [21]. Ми починаємо з імпорту деяких важливих бібліотек (numpy і хешів), а потім вже визначаємо функцію, яка буде генерувати наш ключ підпису.

Як ми бачили вище в описі OTS, ключ підпису залежить від повідомлення. Якщо повідомлення має довжину k бітів, то ключ підпису буде матрицею розміру $2 \times k$. Таким чином створюються два секретних ключа sk_1 і sk_2. Функція ключа підпису приймає повідомлення (яке має бути бажано у формі рядка, але перетворюється на нього лише для підтвердження) і перетворює його на послідовність бітів. На наступному кроці бітовий рядок повідомлення перетворюється на список бітів і зрештою використовується для створення матриці як масиву numpy. Наведемо програмну реалізацію функції генерації пари секретних ключів:

```
def signing_key(message):
    message = str(message)
    message =  ''.join(format(ord(i), '08b') for i in message)
    message = [int(i) for i in message]
    k = len(message) # Security Parameter
    return np.random.randint(0,2,size=(2*k,k)).tolist()
```

Зауважимо, що ця функція створює ключі, які мають довжину таку ж, як і довжина повідомлення. Насправді зазвичай повідомлення хешують, тому довжина ключа - параметр хеш-функції. В цьому коді функція str() просто повертає рядок, join() поєднує з роздільником, ord() повертає unicode, метод format() повертає форматоване значення на основі вказаного засобу форматування 08b (08 – мінімум вісім біт) та метод tolist() перетворює масив NumPy на список Python, не змінюючи його даних або розмірів. Таким чином, основна задача цієї функції складається в тому. щоб визначити довжину повідомлення, використовуючи заповнити яку відповідну матрицю випадковими числами. Отже, не можна використовувати хешування, а на цьому етапі застосувати який-небудь простий метод для вибору шматочків довільної довжини з початкового повідомлення.

Тепер, коли у нас є ключ підпису, ми можемо легко створити ключ перевірки шляхом побудови функції verifying_key(key), яка виводить ключ верифікації. Після підпису повідомлення signature(message,key), будуємо функцію verification(message,verification_key,signature), яка приймає три параметра. Далі вводимо повідомлення, включаємо функції та отримуємо позитивний результат.

```
message = "Shine On You Crazy Diamond."
key=signing_key(message)
verification_key=verifying_key(key)
signature=signature(message,key)
verification(message,verification_key,signature)
```

```
OUTPUT: True
```

Найбільшою проблемою OTS Лампорта (крім того факту, що його можна використовувати лише один раз) є залежність ключа підпису від довжини повідомлення. По-перше, повідомлення не завжди мають однаковий розмір. Деякі повідомлення можуть складатися лише з кількох бітів, а деякі – це послідовності з мільйонів бітів. Таким чином, алгоритм не може бути дійсно стандартизований для одного конкретного розміру ключа. Друга проблема полягає в тому, що розмір ключа підпису збільшується разом зі збільшенням розміру повідомлення. Тому OTS Лампорта не просунувся далеко у світі практичних криптографічних додатків. Тим не менш, ця схема є фундаментом в дереві Меркла, яке використовується, щоб принаймні вирішити проблему одноразового використання.

3.2 Багаторазовий підпис на базі дерева Меркла

Ми вже розглядали схему підпису на дереві Меркла у розділі 2.2.1. Зараз настав час для імплементації цієї схеми в коді, тому основні моменти, особливо важливі при написанні коду, необхідно ще раз згадати, наголошуючи на задачі імплементації. Отже, щоб створити підпис повідомлення, Аліса (див. рисунок 1.2).

вибирає пару відкритих/приватних ключів Лампорта, яка ще не використовувалася, і створює підпис Лампорта, пов'язаний із повідомленням. На рисунку 3.1 нижче обрана пара 3. Далі Аліса шукає шлях автентифікації, який складається з вузлів, необхідних для обчислення кореня Меркла, знаючи початковий вузол. Аліса передає Бобу номер пари ключів Лампорта, повідомлення та підпис Меркла, що складається з:

- Підпис Лемпорта (8 КіБ).
- Відкритий ключ Лампорта t, який використовувався для генерації підпису (16 КіБ).
- Шлях автентифікації (256 біт * 3 для дерева на рисунку 3.1)

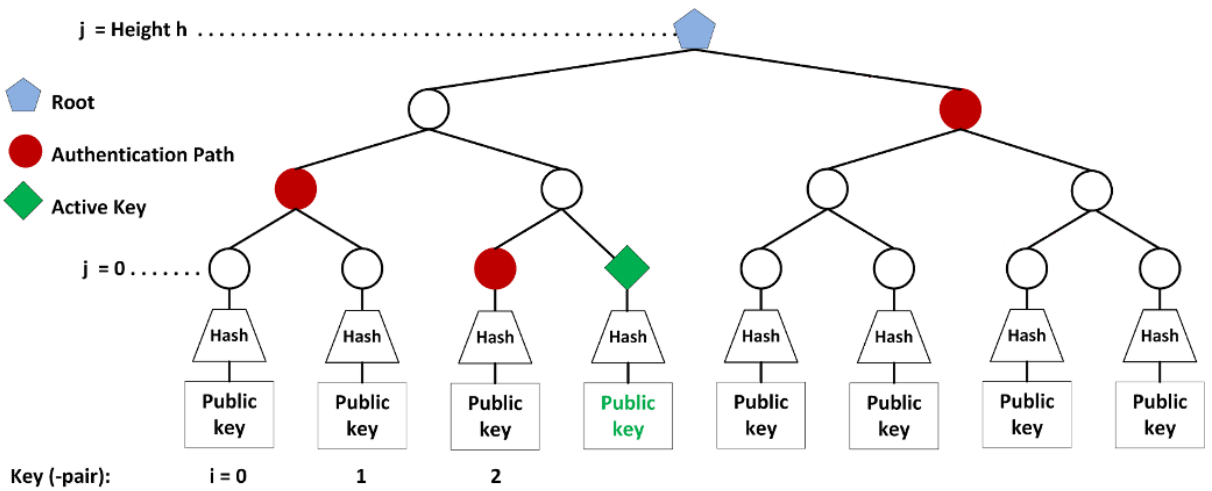

Рисунок 3.1 – Шлях автентифікації (рисунок створено самостійно)

Таким чином, розмір підпису Merkle становить приблизно 24 КіБ.

Боб хоче знати, чи повідомлення є справжнім і чи не було воно змінено під час передачі. Він починає з перевірки правильності підпису Лемпорт. Якщо це правильно, Боб реконструює дерево, використовуючи шлях автентифікації, і порівнює отриманий корінь Меркла з відкритим ключем, який Аліса відправила йому раніше. Якщо вони рівні, підпис є дійсним.

Таким чином, дерево Меркла дає змогу об'єднати кілька відкритих ключів лампорта в один відкритий ключ для багаторазового використання та набагато

меншого розміру, ніж відкритий ключ Лампорта. Його відносно просто реалізувати, оскільки він вимагає лише базових структур даних і хеш-функцій, які вже реалізовані в більшості мов. Однак для забезпечення надійності системи необхідно використовувати достатньо надійний генератор випадкових чисел. Однак використання хеш-дерев передбачає багато чисельних обчислень для створення хешів, а також більшого розміру підпис. Порівняння двох схем підписів наведено у таблиці 3.1.

Таблиця 3.1 – Порівняння схем підписів (таблиця виконана самостійно)

Схема підпису	**Розмір відкритого ключа**	**Розмір закритого ключа**	**Розмір підпису**
Меркл (N листків)	256 біт	2*N*16 КіБ	8 КіБ + 16 КіБ + log2(N)*256 біт
Лампорт	16 КіБ	16 КіБ	8 КіБ

Реалізація алгоритму проходить на мові Python, тому що цей алгоритм відносно легкий з точки зору необхідних ресурсів, обчислювальної потужності Python більш ніж достатньо.

Щоб реалізувати алгоритм підпису на основі дерев Меркла, треба створити класи MerkleTree, LamportSignature, а також, по необхідності, класи Client і Server для частини моделювання мережі.

Клас MerkleTree має атрибути дерева, n_levels і n_leaves. Перший представляє дерево у формі словника кортежів (позиція, дані), причому сама позиція є кортежем (рівень, індекс). Ми рахуємо в дереві знизу вгору та зліва направо, тому корінь знаходиться в позиції (level_max, 0). Висоту дерева представляє n_levels (ми рахуємо від 0, тому простий корінь має висоту 0). Нарешті, n_leaves представляє кількість листя на дереві. Перерахуємо методи цього класу:

- add_node(data, position, hashed) дозволяє вам додати вузол зі значенням даних у позицію дерева дерева. Модульний тест: перевіряє, чи функція перетворює дані, якщо необхідно, і чи вставляє їх у потрібне місце в дереві.

- generate_tree() дозволяє генерувати все дерево з листя. Модульний тест: перевіряє правильну генерацію певного дерева.
- get_root() дає нам корінь дерева. Модульний тест: перевіряє корінь даного дерева.
- get_brother_node_hash(position) повертає хеш сусіднього вузла. Модульний тест: перевіряє правильність роботи на заданому прикладі.
- get_brother_node_position(position) повертає позицію сусіднього вузла. Тест пристрою: перевіряє роботу незалежно від положення
- get_authentication_path(index) повертає шлях автентифікації, починаючи з аркуша індексу. Модульний тест: перевіряє правильність роботи на конкретному прикладі
- get_authentication_path_hashes(index) повертає хеші шляху автентифікації, починаючи з аркуша індексу.
- hash(data) повертає хеш даних.

Клас LamportSignature має атрибути private_key, public_key і used. Перші два містять приватний і відкритий ключі, використані є логічним значенням, яке вказує, чи підпис уже використовувався. Два ключі містять 256 пар по 32 байти, тобто 256 біт: кожен ключ містить 2x256x256 біт або 16 КіБ. Під час створення об’єкта LamportSignature генерується випадковий закритий ключ, а з нього відповідний відкритий ключ. Методи цього класу такі:

- generate_private_key() генерує випадковий ключ. Модульний тест: перевіряє довжину та тип повернутого елемента.
- generate_public_key() генерує відповідний відкритий ключ. Модульний тест: Те саме.
- concatenate_key(key) об’єднує всі пари з 32 байтів, потім усі отримані групи з 64 байтів, щоб нарешті зберегти лише список, що містить усі біти. Модульний тест: перевіряє правильну роботу в конкретному випадку.

- decatenate_key(key) виконує операцію, протилежну попередньому методу. Модульний тест: перевіряє, що об’єднання та декатенація відновлює вихідний ключ.
- get_key(key_type, concatenate) повертає ключ key_type (відкритий чи приватний), об’єднаний чи ні залежно від значення concatenate. Модульний тест: перевіряє правильне функціонування різних сценаріїв.

sign(msg) повертає підпис даного повідомлення у вигляді звичайного тексту.

- verify(msg, signature, public_key) перевіряє підпис за допомогою msg і public_key.

Заключний модульний тест: перевіряє правильність роботи незалежно від того, правдивий чи хибний підпис.

Деревоподібна хеш-структура зараз використовується в багатьох системах, таких як однорангові системи, git і блокчейн. Однак через тривалий час обчислення та великі розміри підпису схема Меркла рідко використовується для підпису базових повідомлень, особливо тому, що вже існують інші алгоритми, такі як RSA.

Зростання інтересу до схеми підпису Меркла викликано тим фактом, що вона, як вважають, стійка до атак квантових алгоритмів. Дійсно, вважається, що більшість систем, які зараз використовуються для підпису або шифрування повідомлень, є вразливими до атак квантового алгоритму Гровера, який, очевидно, може зламати їх без зусиль. Однак ми повинні дивитися на речі в перспективі, тому що всі ці алгоритми застаріють лише тоді, коли квантові комп’ютери будуть демократизаційні, а це ще далеко не так.

4 АНАЛІЗ АЛГОРИТМІВ ОБХОДУ ДЕРЕВА МЕРКЛА

У цьому розділі ми зосередимося на проблемі ефективного обчислення наступного шляху автентифікації, необхідного для схеми підпису Меркла. Проблема обходу дерева Меркла відповідає на питання, як ефективно обчислювати шлях автентифікації для всіх листів один за одним, починаючи з першого. У цій частині ми зосередимося на подібності та відмінності між уже описаними алгоритмами. Наша основна мета полягає в тому, щоб розглянути вимоги до часу та простору для кожної техніки обходу з попереднього розділу та порівняти їх ефективність у різних ситуаціях. Спочатку розглядаються аргументи про класичний обхід дерева та його вдосконалення. Характеристики техніки фрактального обходу будуть надані окремо від двох інших через її абсолютно різні ідеї та структуру. У самому конце цього розділу наведено порівняльний аналіз та приклад роботи трьох алгоритмів для дерева Меркле фіксованих розмірів. Треба пояснити, що в цьому розділі під терміном операція мається на увазі елементарна операція, така як обчислення листкового значення або оцінка хеш-функції.

4.1 Класичний обхід

Класичний обхід дерева Меркла базується на алгоритмі TREEHASH для більш ефективного обчислення вузлів дерева. Для дерева максимальної висоти H ми запускаємо один екземпляр TREEHASH для кожної висоти h ($0 \leq h < H$), і таким чином H різних екземплярів оновлюються паралельно на кожному раунді алгоритму. На висоті h ми маємо рівно 2^{H-h-1} правих вузлів і стільки ж лівих. Звичайно ми використовуємо TREEHASH для обчислення значень цих вузлів, і, логічно, кількість необхідних операцій залежить від h. Для вузла висотою h необхідна кількість операцій TREEHASH рівно $2^{h+1}-1$. Таким чином, загальну кількість обчислень, необхідних для певної висоти h, можна округлити до

$2^{H+1} = 2N$, що рівно 2 обчислення на раунд (це причина, чому кожен екземпляр TREEHASH оновлюється двома одиницями обчислення на кожному раунді). Якщо ми додамо необхідні операції для всіх висот h, ми отримаємо загальну кількість необхідних операцій на раунд класичного алгоритму, яка становить 2H = 2log(N) операцій.

Говорячи про вимоги до зберігання класичного алгоритму, ми наведемо наступні аргументи. Як уже пояснювалося, ми можемо мати до H одночасно запущених процесів TREEHASH (по одному для кожної висоти h). Для заданого h ми спостерігаємо, що стек на цій висоті може містити до h+1 значень. Це означає, що після кожного раунду алгоритму максимум 1+2+...+H значень зберігається в стеках. Таким чином, ми оцінюємо просторову складність алгоритму як $O(H^2/2) = O(\log^2 N/2)$. Уявіть, що нам дали достатньо велике дерево Меркла, і ми застосували до нього класичний обхід. Чим більша висота екземпляра TREEHASH, тим більша кількість збережених значень вузлів у його стеку, що призводить до великих вимог до пам'яті та робить класичний обхід дуже неефективним для великих дерев. Класичний алгоритм обходу дерева Меркла виглядає так:

- Встановіть *leaf* = 0.
- Вихід:
    - Обчислення та виведення *leaf* за допомогою LEAFCALC(*leaf*)
    - Для кожного $h \in [0, H-1]$ h виведіть $\{ auth_h \}$.
- Оновіть вузли авторизації:

Для h такого, що 2^h ділить *leaf* + 1:

- Встановити $auth_h$ як єдине значення вузла в $stack_h$.
- Встановити початковий вузол = $(leaf + 1 + 2^h) \oplus 2^h$.
- *.initialize(startnode, h).*
- Створення стеків:

- Для всіх $h \in [0, H-1]$:
- $stack_h$ *.update(2).*

- Петля
  - Встановити *leaf* = *leaf* + 1.
  - Якщо *leaf* < 2^H, перейдіть до кроку 2.

4.2 Вдосконалення класичного алгоритму обходу

У [22] представлено вдосконалення класичного алгоритму обходу Меркла. Основна мета цього вдосконаленого алгоритму – зменшити вимоги до пам'яті. У класичному алгоритмі до H екземплярів хешу дерева можуть бути одночасно активними, по одному для кожної висоти, меншої за H. В одному хеші дерева одночасно має зберігатися до h+1 вузлів. Отже, до

$$\sum_{h=0}^{H-1} h+1 = H(H+1)/2$$

вузли повинні зберігатися протягом однієї генерації підпису. Основна ідея вдосконаленого алгоритму полягає в тому, щоб зменшити вимоги до пам'яті шляхом зменшення кількості активних екземплярів хешу дерева під час генерації підпису.

Щоб створити наступний вузол автентифікації в stackh, потрібно $2^{h+1}-1$ операцій. Щоб створити наступний вузол автентифікації в $stack_{h+1}$, потрібно $2^{h+2}-1$ операцій. У класичному алгоритмі протягом однієї генерації підпису $stack_h$.update(2) і $stack_{h+1}$.update(2) викликаються один раз. Але якщо для перших $2^h/2$ підписів викликається лише $stack_h$.update(4), то $stack_h$ буде обчислено після $2^h/2$ підписів. Для наступних $2^h/2$ підписів можна викликати

$stack_{h+1}$.update(4), щоб наприкінці, після 2^h підписів, були обчислені ті самі значення, що й у класичному алгоритмі. Однак у перших 2^h /2 підписах $stack_{h+1}$ було порожній, а в наступні 2^h /2 підписів лише один вузол зберігався в $stack_h$. Отже, ми можемо зменшити вимоги до пам'яті. Алгоритм обходу дерева Меркла в цьому випадку виглядає так:

- Встановіть *leaf* = 0.
- Вихід:
  - Обчислення та виведення *leaf* за допомогою LEAFCALC(*leaf*)
  - Для кожного $h \in [0, H-1]$ h виведіть $\{auth_h\}$.
- Оновіть вузли авторизації:

Для h такого, що 2^h ділить *leaf* + 1:

- Встановити $auth_h$ як єдине значення вузла в $stack_h$.
- Встановити початковий вузол = (*leaf* + 1 + 2^h) $\oplus$ 2^h.
- $stack_h$.*initialize(startnode, h).*

- Створення стеків:

Повторити наступне $H-1$ раз:

- Нехай l_{min} буде мінімумом $\{stack_h$ `.low} для всіх h=0, .,H – l.`
- Вибираємо h таке t, щоб $stack_h$ .low = l_{min}.
- $stack_h$.*update(2).*

- Петля
  - Встановити *leaf* = *leaf* + 1.
  - Якщо *leaf* < 2^H, перейдіть до кроку 2.

Алгоритм вважає за краще завершити $stack_h$ для найнижчого h, якщо інший стек не має кінцевий вузол нижче. Отже, алгоритм не почне обчислювати новий

стек, доки в жодному стеку не буде збережено жодного вузла, висота якого нижча за h.

Представлений алгоритм просто зберігає 3log(N) вузлів у гіршому випадку та потребує обчислення 2log(N) операцій у гіршому випадку. Причому N є кількістю доступних підписів, отже $N = 2^n$. В подальшому було представлено алгоритм, який базується на тій самій ідеї, але досягає цього лише за допомогою log(N) операцій і 3log(N) збережених вузлів. Це завдяки кращому, але складнішому алгоритму планування.

4.3 Фрактальне представлення дерева Меркла

Основна ідея фрактального обходу [23] полягає в тому, щоб розділити дерево Меркла на піддерева, а також зберегти та обчислити ці піддерева замість окремих вузлів. Кожне піддерево t має однакову висоту h. Кожен корінь одного піддерева підписується підписом вищого піддерева. Існують L = H/h рівні піддерев від низу до верху дерева Меркла. Якщо вузол $a_{i*h,j}$ є листом піддерева ti, то i позначає рівень піддерева, де $1 \leq i \leq L$. Ми визначаємо піддерево $t_{i,j}$ як j-те піддерево зліва від рівня i.

Під час генерації ключа зберігаються всі піддерева Exist_i, які містять вузол наступного шляху аутентифікації. На початку ці піддерева Exist_i є крайніми лівими піддеревами $\text{Exist}_i = t_{i,0}$ кожного рівня i = 1, ...,L. Якщо $a_{i*h,j}$ є коренем піддерева Exist_i, то наступним піддеревом рівня i, яке буде потрібно, є піддерево Desire_i з коренем $a_{i*h,j+1}$. Ці піддерева Desire_i також зберігаються під час генерації ключа.

Етап створення підпису знову складається з двох етапів. На етапі виведення виводяться підпис і шлях автентифікації. На етапі оновлення встановлюються

піддерева з наступним вузлом автентифікації $\mathrm{Exist_i}$ та обчислюються піддерева $\mathrm{Desire_i}$. Для генерації піддерева $\mathrm{Desire_i}$ ми використовуємо дещо модифікований алгоритм хешування дерева. У цьому алгоритмі хешування дерева ми зберігаємо всі вузли з висотою, меншою за i*h, оскільки ці вузли є вузлами піддерева. Ми також зупиняємося на крок раніше, тому що нам не потрібно обчислювати корінь (замість кореня лист піддерева рівня $i+1$ використовується як вузол автентифікації). Далі приводимо алгоритм фрактального обходу дерева Меркла:

- Встановіть *leaf* = 0.
- Вихід:
  - Обчислення та виведення *leaf* за допомогою LEAFCALC(*leaf*)
  - Для кожного $h \in [0, H-1]$ h виведіть $\{\mathrm{auth_h}\}$.
- Наступне піддерево:

Для кожного i, для якого $\mathrm{Exist_i}$ більше не потрібен, тобто для i $\in$ {1, 2, ...,L} із *leaf* = 1(mod 2^{hi}):

- Встановити $\mathrm{Exist_i} = \mathrm{Desire_i}$.
- Створіть новий пустий $\mathrm{Desire_i}$ (якщо *leaf* + $2^{hi} < 2^H$).

- Зростання субдерев
  - Для кожного i $\in$ {1, 2, ..., h}: виростіть дерево $\mathrm{Desire_i}$, застосувавши 2 одиниці до модифікованого хешу дерева (якщо $\mathrm{Desire_i}$ не завершено).

Збільште *leaf* і поверніться до кроку 2 (поки лист < 2^H).

Таким чином, алгоритм виконує до двох операцій для кожного піддерева в кожному раунді. Загалом є L = H/h піддерев. Це дає верхню межу часу обчислення $\mathrm{sig_{time}} = 2(L - 1) \leq 2H/h$. Максимальний розмір графу, необхідний під час

обчислення, — це sig_{space}. Існує L існуючих піддерев і L−1 бажаних піддерев. Кожне дерево складається з $2^{h+1} - 2$ вузлів, оскільки значення кореня не повинно зберігатися. Під час обчислення потрібного дерева необхідно зберегти до h*(i−1) хвостових вузлів. Отже, $sig_{space} \leq (2L - 1)(2^{h+1} - 2) + h(L - 2)(L - 1)/2$.

Хорошим значенням для h, коли вимоги до простору є мінімальними, буде h =logH = loglogN. Використання цього параметра призведе до обмеження часу та простору: $sig_{time} = 2logN/loglogN$ та $sig_{space} = 5/2 \log^2 N /loglogN$.

Вимоги до простору можна ще трохи зменшити шляхом незначної модифікації. Ідея полягає в тому, що фактично майже у всіх випадках ми можемо відкинути проміжні значення існуючих піддерев, як тільки вони увійшли до відповідного потрібного піддерева. Таким чином, вимоги до простору зменшуються до 1,5log2 (N)/2log log(N). Ці результати покращують вимоги до часу та простору для всіх раніше опублікованих методів обходу, але для більших дерев Меркла ми все ще маємо велике споживання простору, що робить фрактальну техніку більш не ефективною, ніж інші обходи.

4.4 Порівняння алгоритмів

Щоб бути конкретним і дослідити поведінку трьох методів, ми наведемо кілька прикладів того, як вони працюватимуть у реальному житті, використовуючи три типові дерева. Ці дерева Меркла мають різну висоту і дають уявлення про те, в якій саме ситуації ми перебуваємо можна використовувати алгоритми. Нехай T_i (i =1,2,3) — дерева Меркла висотою 1 H =5, 2 H =10 і 3 H = 20. Вимоги до часу для цих дерев (у кількості елементарних операцій) наведені в наступній таблиці 4.1:

Таблиця 4.1 – Вимоги до часу в кількості елементарних операцій (таблиця виконана самостійно)

Техніка обходу	T_1	T_2	T_3
Класичний Меркл	10	20	40
Вдосконалений	5	10	20
Фрактальний	4.3068	6.0207	9.2552

Тепер ми покажемо вимоги до простору (у кількості збережених хеш-значень) для тих самих дерев Меркла. Більш різкі відмінності можна побачити в наступній таблиці 4.2:

Таблиця 4.2 – Вимоги до простору в кількості елементарних операцій (таблиця виконана самостійно)

Техніка обходу	Вимоги до простору для T_1	Вимоги до простору для T_2	Вимоги до простору для T_3
Класичний Меркл	12.50	50	200
Вдосконалений	15	30	60
Фрактальний	8.0753	22.5774	69.4138

Говорячи про вимоги до простору, ми повинні зауважити той факт, що реальні вимоги до простору для всіх алгоритмів залежать від довжини виводу нашої хеш-функції. Наступні та останні дві таблиці показують нам вимоги до простору в бітах для хеш-функцій із 160-бітним і 256-бітним виводом:

Таблиця 4.3 – Вимоги до простору для 160-бітної хеш-функції (таблиця виконана самостійно)

Техніка обходу	Вимоги до простору для T_1 (кбіт)	Вимоги до простору для T_2 (кбіт)	Вимоги до простору для T_3 (кбіт)
Класичний Меркл	0.250	1	4

Вдосконалений	0.3	0.6	1.2
Фрактальний	0.1615	0.4515	1.3883

Таблиця 4.4 – Вимоги до простору для 256-бітної хеш-функції (таблиця виконана самостійно)

Техніка обходу	Вимоги до простору для T_1 (кбіт)	Вимоги до простору для T_2 (кбіт)	Вимоги до простору для T_3 (кбіт)
Класичний Меркл	0.250	1.6	6.4
Вдосконалений	0.48	0.96	1.92
Фрактальний	0.2584	0.7225	2.2212

Результати нашого порівняння легко побачити в наведених таблицях і помітити, що фрактальний алгоритм є найшвидшим, але не завжди найбільш економічним. Для більших дерев можливі менші потреби в просторі за допомогою логарифмічного обходу. Такі висновки можна зробити для кожної конкретної ситуації, залежно від того, які наші потреби та скільки часу та місця ми можемо витратити. Таким чином, загальні висновки не є гарною ідеєю, кращими є висновки для окремих випадків [24].

ВИСНОВКИ

У роботі проведено аналіз шляхів підвищення ефективності побудови сучасних постквантових алгоритмів цифрових підписів. Було проведено огляд різних напрямів досліджень, які вивчалися в постквантовій криптографії, але перед цим розглядаються алгоритми та методи квантових обчислень, щоб зрозуміти, які загрози вони несуть безпеці криптографічних схем. Розглянуто схеми цифрового підпису, проведено порівняльний аналіз постквантових алгоритмів. На основі цього аналізу виявлено недоліки та переваги схем підпису. Постійно зростаюча загроза великомасштабного квантового комп'ютера відновила інтерес до цієї галузі, і за останнє десятиліття відбувся значний розвиток. Показано життєздатність підписів на основі хешування на деревах Меркла, які згодом привели до створення схем підписіва із сімейства SPHINCS , в яких не потрібно постійно підтримувати стан системи.

Проведено аналіз різноманітних схем підписів на основі хеша. Детально розглянуто алгоритм OTS Лампорта, який лежить в основі більшості постквантових схем підписів. Розглянуто алгоритм WOTS, приписуваний Вінтерніцу, який дозволяє зменшити розмір підпису за рахунок додаткового часу виконання. Детально вивчені схеми багаторазових підписів, які реалізовані на деревах Меркла. Відзначено, що основні проблеми сучасних схем підписів такі ж, як і проблеми, які зустрічаються при розгляді дерева Меркла. Це перш за все проблема аутентифікації шляхів і проблема обходу. Для більш глибокого вивчення алгоритмів цифрових підписів була розроблена робоча модель ОТС Лампорта, яка була реалізована в коді мовою Python. На основі цього моделювалося дерево Меркла для отримання багаторазових підписів. Аналізувався розмір відкритих і закритих ключів, розмір самого підпису при різних вхідних параметрах.

Далі вивчалась проблема ефективного обчислення наступного шляху автентифікації, необхідного для схеми підпису Меркла. Проблема обходу дерева

Меркла відповідає на питання, як ефективно обчислювати шлях автентифікації для всіх листів один за одним, починаючи з першого. В цьому ракурсі проведен аналіз подібності та відмінності між алгоритмами підпису, було розглянуто вимоги до часу та простору для кожної техніки обходу з порівнянням їх ефективність у різних ситуаціях. Розглянуті аргументи про класичний обхід дерева та його вдосконалення. Ймовірно, що з вирішенням проблеми ефективного обходу дерева Меркла доведуть свою корисність стохастичні методи теорії ігор [25]. Наведено порівняльний аналіз роботи трьох алгоритмів для дерева Меркла фіксованих розмірів.

На основі моделювання та вивчення запропонованих схем підписів у цій роботі запропоновано конкретну реалізацію системи на основі дерева Меркла. Розроблено працюючі коди схем підписів. Результати дослідження мають значення для систем постквантових алгоритмів підписів. По-перше, дослідження показує, що структури на кшталт дерева Меркла є найбільш перспективними кандидатами для створення ефективного шаблону постквантових підписів. По-друге, дослідження підкреслює важливість вирішення ключових проблем, пов’язаних з шляхом автентифікації та проблемою ефективного обходу дерев.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Stinson, Douglas R. *Cryptography: theory and practice*. Chapman and Hall/CRC, 2005.
2. Mermin, N. David. *Quantum computer science: an introduction*. Cambridge University Press, 2007.
3. Shor, Peter W. "Algorithms for quantum computation: discrete logarithms and factoring." *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994. P. 124-134.
4. Proos, John, and Christof Zalka. "Shor's discrete logarithm quantum algorithm for elliptic curves." *arXiv preprint quant-ph/0301141* (2003).
5. Grover, Love K. "A fast quantum mechanical algorithm for database search". Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery. 1996. P. 212–219.
6. Merkle, Ralph C. "A certified digital signature." *Conference on the Theory and Application of Cryptology*. New York, NY: Springer New York, 1989. P. 218-238.
7. Lamport, Leslie. "Constructing digital signatures from a one way function." Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, Oct. 1979.
8. Butin, Denis. "Hash-based signatures: State of play." *IEEE security & privacy* 15.4 (2017): 37-43.
9. Bernstein, D.J.; Hülsing, A.; Kölbl, S.; Niederhagen, R.; Rijneveld, J.; Schwabe, P. The SPHINCS+ Signature Framework. URL: http://www.informationweek.com/news/201202317. (дата звернення: 4.05. 2024).
10. McEliece, Robert J. "A public-key cryptosystem based on algebraic." *Coding Thv* 4244 (1978): 114-116.
11. Ding, Jintai, and Albrecht Petzoldt. "Current state of multivariate cryptography." *IEEE Security & Privacy* 15.4 (2017): 28-36.

12. Ducas, Léo, et al. "Lattice signatures and bimodal Gaussians." *Annual Cryptology Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.
13. Ducas, Léo, et al. "Crystals–dilithium: Digital signatures from module lattices. 2018." URL: https://eprint.iacr.org/2017/633 (дата звернення: 30.03.2024).
14. Fouque, P.A et al. Fast-Fourier Lattice-Based Compact Signatures over NTRU. URL: https://www.di.ens.fr/~prest/Publications/falcon.pdf (дата звернення: 5.04.2024).
15. Jao, David, and Luca De Feo. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies." *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4*. Springer Berlin Heidelberg, 2011. Aggarwal C. C. Recommender systems. Cham : Springer International Publishing, 2016. P.518.
16. Costello, Craig, Patrick Longa, and Michael Naehrig. "Efficient algorithms for supersingular isogeny Diffie-Hellman." *Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I 36*. Springer Berlin Heidelberg, 2016.
17. Goldwasser, Shafi, Silvio Micali, and Ronald L. Rivest. "A digital signature scheme secure against adaptive chosen-message attacks." *SIAM Journal on computing* 17.2 (1988): 281-308.
18. Tan, Teik Guan, Pawel Szalachowski, and Jianying Zhou. "Challenges of post-quantum digital signing in real-world applications: A survey." *International Journal of Information Security* 21.4 (2022): 937-952.
19. Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. “XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions.” In: Post-Quantum Cryptography – PQCrypto 2011. Ed. by Bo-Yin Yang. Vol. 7071. LNCS. Springer, 2011, pp. 117–129.

20. Srivastava, Vikas, Anubhab Baksi, and Sumit Kumar Debnath. "An overview of hash based signatures." *Cryptology ePrint Archive* (2023). URL: https://eprint.iacr.org/2023/411

21. Cornelius Schätz. "The Lamport-Diffie One Time Signature Scheme in Python" URL: https://medium.com/coinmonks/the-lamport-diffie-one-time-signature-scheme-in-python-9066084f1f06 (дата звернення: 5.04.2024).

22. Szydlo, Michael. "Merkle tree traversal in log space and time." *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*. Springer Berlin Heidelberg, 2004.

23. Jakobsson, Markus, et al. "Fractal Merkle tree representation and traversal." *Topics in Cryptology—CT-RSA 2003: The Cryptographers' Track at the RSA Conference 2003 San Francisco, CA, USA, April 13–17, 2003 Proceedings*. Springer Berlin Heidelberg, 2003.

24. Wang, Xinyue, et al. "Integrating Merkle Trees with Transformer Networks for Secure Financial Computation." *Applied Sciences* 14.4 (2024): 1386.

25. Vlasenko, L. A., et al. "Stochastic Descriptor Pursuit Game." *Cybernetics and Systems Analysis* (2024): 1-9.

26. Kachko, O. G., et al. "Hash-based cryptography, its security and feasibility in modern cryptosystems" (2024): 1-9.

27. Kachko, O. G., et al. "Substantiation and proposals for the selection, improvement and standardization of the post-quantum electronic signature mechanism at the national and international levels" *Methods of promising cryptographic transformations* (2021): 5-7.

28. Kachko, O. G., et al. "Basic principles and results of comparison of electronic signatures properties of the postquantum period based on algebraic lattices" *Methods and algorithms of cryptographic information protection* (2021): 1-17.