УДК 004.056.5
DOI: https://doi.org/10.17721/1812-5409.2026/1.24

Олексій БАШУК, асп.
ORCID ID: 0009-0002-6778-2943
e-mail: a.bashuk@knu.ua
Київський національний університет імені Тараса Шевченка, Київ, Україна

ПОРІВНЯЛЬНИЙ АНАЛІЗ ПОСТКВАНТОВИХ ПРОТОКОЛІВ ЦИФРОВОГО ПІДПИСУ, БАЗОВАНИХ НА ГЕШ-ФУНКЦІЯХ

В епоху активного розвитку інформаційних технологій і пристроїв поява портативних та загальнозастосовних квантових комп'ютерів є лише питанням часу. І хоч користь від їхньої в рази більшої ефективності, порівняно зі звичайними бітовими комп'ютерами, буде очевидною, водночас вони принесуть і багато вразливостей до сучасних алгоритмів криптографії та кібербезпеки. Щоб запобігти цьому, фахівці вже розпочали роботу над дослідженнями постквантових протоколів замість сучасних.

Не стали винятком і протоколи цифрового підпису. Наприклад, одним із підходів розв'язання окресленої проблеми є побудова постквантових протоколів цифрового підпису, базованих на геш-функціях. Першими варіантами стали схеми одноразових підписів, базованих на геш-функціях, які були захищені від можливостей квантових комп'ютерів, проте мали очевидну вади одноразовості. Найвідомішим вважають так званий одноразовий підпис Лампорта. Однак цей недолік одноразовості усунули, запропонувавши різні схеми багаторазових цифрових підписів, базованих на створенні та використанні багатьох одноразових підписів. Були запропоновані ланцюгоподібні та деревоподібні схеми багаторазових підписів. Саме реалізації та порівнянню різних схем багаторазового підпису і присвячено пропоновану роботу.

У ході досліджень наведено теоретичні оцінки використання пам'яті та часу, необхідні для роботи різних схем. Подальша реалізація цих схем й остаточні заміри виявили відповідність теоретичних оцінок емпіричним результатам, а також явно продемонстрували переваги та недоліки різних схем. Найкращу продуктивність, зокрема, показали деревоподібні схеми, особливо ті, у яких дерево будується послідовно. Проте в певних випадках має сенс використовувати дерево з повним заглибленням під час побудови.

З огляду на відносну новизну та малу кількість досліджень у цій галузі, багаторазові цифрові підписи мають великий простір для подальших досліджень, наприклад, розгляд N-арних дерев та їхнього порівняння з бінарними.

Ключові слова: ***геш-функція, цифрові підписи, постквантові протоколи, одноразові цифрові підписи, багаторазові цифрові підписи, ланцюгоподібні схеми, деревоподібні схеми.***

Класифікація відповідно до AMS 2020: 81P94, 94A60.

Вступ

Технологічний прогрес не стоїть на місці, і в майбутньому одним із ключових чинників у ньому стануть квантові комп'ютери. Проте до найбільших ризиків їхнього впровадження належить вразливість поточних криптографічних алгоритмів, в основі яких лежить розрахунок обмеженості звичайних бітових комп'ютерів. Традиційні криптосистеми спираються на задачі факторизації цілих чисел або задачі дискретного логарифмування, які легко можна розв'язати на достатньо великих квантових комп'ютерах. З огляду на це останнім часом стали дуже актуальними дослідження у сфері постквантових протоколів у криптографії, що є незалежними від квантових обчислень, а тому стійкими до квантових атак. Протоколи, засновані на геш-функціях, можуть бути стійкими до квантових обчислень.

Не є винятком і цифрові підписи. Найтиповішим прикладом постквантового цифрового підпису став одноразовий підпис Лампорта (Boneh, & Shoup, 2017). Іншими словами, надійність цього підпису безпосередньо випливає з його одноразовості. Ідейно алгоритм полягає у такому. Будується приватний ключ, до якого входять пари випадкових чисел на кожен біт вхідного повідомлення. Для побудови ж публічного ключа обирають певну геш-функцію H, та з її допомогою знаходять геш-значення кожного елемента приватного ключа. Підписом повідомлення за допомогою такого ключа буде множина значень в обсязі кількості бітів вхідного повідомлення, що обираються з кожної пари приватного ключа залежно від значення відповідного біта вхідного повідомлення. За такої побудови легко переконатися, що це дійсно є підписом повідомлення, прогешувавши значення за допомогою функції H, та звіривши з відповідними значеннями публічного ключа. Водночас, не знаючи другої половини приватного ключа, ніхто інший не зможе в майбутньому скористатися ним для підпису повідомлень, відмінних від уже підписаного. Є й інші одноразові підписи, такі як WOTS (Merkle, 1989), WOTS+ (Hülsing, 2013), Bleichenbacher-Maurer OTS (Bleichenbacher, & Maurer, 1994), BiBa OTS (Perrig, 2001) та HORS (Reyzin, L, & Reyzin, N, 2002).

Проте в цих підписів все ще є недолік – вони одноразові. Напрошується логічне бажання – об'єднати багато ключів одноразових підписів в один. З огляду на це був запропонований підпис Меркеля (Boneh, & Shoup, 2017) на основі дерева Меркеля (Katz, & Lindell, 2014, с. 183–184). Створюється певна кількість одноразових ключів і на їхніх публічних ключах, як на листках, будується бінарне дерево, де кожна вершина містить геш-значення її дітей, а значення в корені – публічний ключ такого підпису. Тоді для підписання щоразу використовують черговий одноразовий підпис, що ще не був використаний, а також з ним надають усі необхідні значення з дерева на шляху від відповідного листка, яким було підписано повідомлення, до кореня дерева. Тоді для верифікації перевіряють безпосередньо сам одноразовий підпис, а також те, що надані значення дійсно є елементами дерева, що відповідає публічному ключу кореня.

І хоч такий підхід забезпечує багаторазовість, він все ще є обмеженим. Одним із виходів стали так звані ланцюгові підписи (Chain-Based Signature) (Katz, & Lindell, 2014, p. 465–468). У них теж за основу беруть певний одноразовий підпис. Під час підписання повідомлення підписують не тільки саме повідомлення, але також публічний ключ нового свіжостореного одноразового підпису. Під час наступного підписання вже використовують цей новий підпис, за допомогою якого підписують нове повідомлення, а також публічний ключ ще одного створеного підпису. Таким чином вибудовується ланцюг із підписів. Щоб можна було верифікувати будь-який з них, потрібно надати весь ланцюг разом

ISSN 1728-3817

з відповідними публічними ключами та вхідними повідомленнями, що були підписані від початкового підпису до підпису поточного повідомлення. Разом з початковим публічним ключем, легко переконатися в дійсності кожного з підписів в ланцюзі, зокрема і в тому, що цікавить перевіряльника. Тобто фактично "підписом" повідомлення вважають не безпосередньо зроблений у ланцюзі останній підпис, а весь поточний стан ланцюга разом з останнім підписом. У такого підходу є очевидний недолік – з кожним новим підписом зростає як розмір самого підпису, так і час, необхідний на його верифікацію.

Однак ідея такого підходу давала можливість використання багаторазових цифрових підписів на основі геш-функцій і, як наслідок, привела до реалізації різного роду комбінацій і надбудов. Наприклад, був запропонований підпис на основі HMAC (Hash-based Message Authentication Code) (Lizama-Pérez, 2022). У протоколі цього підпису було використано одноразовий цифровий підпис WOTS, де, попри його одноразовість, він певним чином використовувався двічі – для підписання поточного основного повідомлення та верифікації наступного, а поверх цього був використаний HMAC – для перевірки цілісності інформації під час комунікації.

Щоб скоротити розмір підпису, було запропоновано поєднання ідеї ланцюгового підпису та дерева Меркеля. Наприклад, для побудови деревоподібного підпису (Tree-Based Signature) (Katz, & Lindell, 2014, p. 468–473) пропонується будувати бінарне дерево, аналогічне дереву Меркеля, тільки кожна вершина містить у собі не геш-значення своїх дочірніх вершин, а їхній підпис. Для цього для кожної дочірньої вершини створюють пари ключів для підпису, якщо вони ще не були створені раніше, і після цього підписують публічні ключі цих дочірніх вершин. Крім того, кожній вершині відповідає деякий рядок "a" з 0 та 1 такий, що в дочірніх вершин поточної вершини будуть рядки "a0" та "a1", а кореню при цьому відповідає порожній рядок. Вершини, у яких довжина рядка дорівнює довжині бітового значення вхідних повідомлень, вважають листками. Однак це дерево не будується повністю одразу, а розбудовується з часом. Публічним ключем такого дерева підписів буде публічний ключ вершини дерева. Під час підписання ж чергового вхідного повідомлення йде спуск по дереву від кореня так, щоб рядок кожної вершини був префіксом бітового запису вхідного повідомлення. Якщо в якоїсь вершини ще немає дітей і вона не є листком, як уже зазначалося, для неї створюються ключі одноразових підписів та підписуються їхні публічні ключі за допомогою приватного ключа поточної вершини. Якщо ж вона є листком, то за допомогою її приватного ключа підписується відповідне вхідне повідомлення. Якщо ж таке вхідне повідомлення вже було підписане раніше – використовується збережене значення. У підсумку, підписом у такому дереві підписів буде безпосередньо сам підпис вхідного повідомлення у листку, усі підписи у вершинах на шляху від листка до кореня, а також публічні ключі відповідних дочірніх вершин. Для верифікації ж просто перевіряють, що всі підписи є дійсними (а отже, і публічні ключі дочірніх вершин теж), а також сам підпис вхідного повідомлення.

За такої побудови, всі підписи будуть мати фіксовану довжину, що відповідатиме довжині бітового запису вхідних повідомлень, а тому, починаючи з деякого моменту, довжина підпису та час, необхідний на його верифікацію, будуть кращими, ніж у підпису з ланцюговою структурою. Проте для зберігання цього дерева буде потрібно більше пам'яті, ніж для зберігання підпису з ланцюговою структурою. Крім цього, на відмінну від дерева Меркеля, немає необхідності в побудові одразу всього дерева, через що і зникає обмеженість явного ліміту кількості одноразових підписів (залишається лише обмеження пристроїв на зберігання повного такого дерева). Також варто зауважити, що за такої структури в підписі вхідних повідомлень не фігурують раніше підписані вхідні повідомлення.

Якщо ж фігурування попередніх вхідних повідомлень у поточному підписі повідомлення не є проблемою, можна будувати подібне альтернативне дерево. Якщо запропоноване вище дерево будується за аналогією до алгоритму пошуку в глибину, добираючись до листків від кореня дерева фіксованого розміру, то можна будувати дерево за аналогією до алгоритму пошуку в ширину, а саме будувати вершини послідовно залежно від їхньої відстані до кореня дерева (Sequential Tree-Based Signature). При кожному новому підписанні вхідного повідомлення обирається наступна в черзі вершина, для неї створюються дві дочірні вершини з новими одноразовими підписами кожна та додаються в кінець черги, і, за допомогою приватного ключа поточної вершини, підписуються публічні ключі новостворених дочірніх вершин разом із вхідним повідомленням. Підписом у такому дереві, відповідно, вважають усі підписи на шляху від поточної вершини до кореня, разом із публічними ключами дочірніх вершин та відповідними минулими вхідними повідомленнями. За такої побудови кількість вершин, що фігурують в одному підписі, буде суттєво меншою, а отже, і кількість верифікацій одноразових підписів буде меншою, що має вплинути як на час підпису та верифікації, так і на розмір одного підпису, порівняно з попереднім запропонованим деревом. З іншого боку, в підписі будуть також фігурувати попередньо підписані вхідні повідомлення, що збільшить розмір одної ланки підпису.

Ще однією альтернативою буде аналогічний підхід до попереднього, але в якому в підписах не будуть використовувати попередньо підписані повідомлення. Натомість кожна вершина матиме не дві дочірні вершини, а три (Wide Sequential Tree-Based Signature). Ліва та права – зберігають свій звичайний функціонал і використовуються в подальшій побудові дерева, тоді як центральна вершина є листочком дерева і використовується лише для підписання поточного повідомлення. Завдяки такій додатковій генерації листка для підписання повідомлень у кожної звичайної вершини дерева, для перевірки проміжних підписів більше не потрібно знати відповідні проміжні повідомлення. Замість них для перевірки проміжних підписів надають відповідні публічні ключі листків. Як наслідок, у підписах не використовуються раніше підписані повідомлення, проте генерується в півтора рази більше одноразових підписів, що впливає як на час, так і на розмір дерева.

Ця робота спрямована на розроблення реалізацій різних видів багаторазових підписів на основі одноразових, а також порівнянню їхньої продуктивності.

1. Базові заміри

Усі вимірювання та дослідження проводилися на базі ноутбуку MacBook Pro 2021 року з процесором Apple M1 Pro та 32 ГБ оперативної пам'яті. Повний код реалізації можна знайти за посиланням: https://github.com/albashuk/Dissertation/tree/master/Experiments/Python.

В основі всіх названих одноразових підписів лежить використання геш-функцій Hash, що повертають HashSize-бітові значення, від вибору яких напряму залежать результати. Для проведення досліджень було обрано та використано одну з найпопулярніших геш-функцій SHA-256 (NIST, 2015). Заміри продуктивності наведені у табл. 1.

Таблиця 1

Заміри функції SHA-256

Кількість ітерацій	Середній час гешування, с	Середній розмір гешу, Байт
1	0.00000715256	152.000
10	0.00000162125	196.000
100	0.00000109196	192.560
1000	0.00000088501	192.792
10000	0.00000094020	192.428
100000	0.00000079224	192.000
1000000	0.00000072344	192.447
10000000	0.00000071610	192.909

2. Теоретична оцінка цифрових підписів

Кожна схема цифрового підпису має три ключові функції: GEN – генерація публічного та приватного ключів підпису, SIGN – генерація цифрового підпису повідомлення за допомогою приватного ключа, VRFY – верифікація цифрового підпису за допомогою відповідного публічного ключа. Саме на часових замірах цих трьох функцій і буде сконцентрована увага. Проте в реальних умовах, у ході підписання повідомлення та верифікації підпису, частину часу займає пересилання підписів через мережу. Ця величина дуже залежить від умов з'єднання та може значно різнитися. Водночас за конкретно заданих умов ця величина лінійно залежить від розміру даних, що передаються. Тому як найбільш оптимальне рішення буде вимірюватися час виконання суто алгоритмічних дій кожної з трьох ключових функцій, а також розмір даних, що пересилаються.

2.1. Лампорт

Для генерації приватного ключа треба створити HashSize пар випадкових r-значень, а для публічного ключа – набір гешів до кожного із геш-значень у приватному ключі. Відповідно часова складність буде:

$$O(T_{GEN}) = HashSize * 2 * O(T_{Hash}) + HashSize * 2 * O(T_{Hash}), \quad (1)$$

а обсяг пам'яті публічного та приватного ключів:

$$O(S_{sk}) = O(S_{pk}) = HashSize * 2 * O(S_{Hash}). \quad (2)$$

Для підписання повідомлення потрібно спершу перевести його в HashSize-бітове значення. Після цього обирають HashSize геш-значень приватного ключа відповідного до бітів значення вхідного повідомлення, а тоді:

$$O(T_{SIGN}) = O(T_{Hash}) + 256 * O(T_1), \quad (3)$$

$$O(S_{SIGN}) = 256 * O(S_{Hash}). \quad (4)$$

Для верифікації підпису потрібно перевести вхідне повідомлення в HashSize-бітове значення, пройти по всіх геш-значеннях підпису, знайти геші від цих геш-значень і порівняти їх із відповідними значеннями публічного ключа відповідно до вхідного повідомлення. Тобто

$$O(T_{VRFY}) = O(T_{Hash}) + HashSize * O(T_{Hash}). \quad (5)$$

Для подальших розрахунків уведемо позначення OTS (One-Time Signature), як схеми одноразового підпису.

2.2. CBS (Chain-Based Signature)

Генерація приватного та публічного ключів повністю відповідають генерації приватного та публічного ключів відповідної схеми OTS. Для підписання нового повідомлення потрібно створити нову пару публічного та приватного ключів OTS. За допомогою останнього приватного ключа в списку необхідно підписати повідомлення разом із новоствореним публічним ключем та додати новостворені ключі разом з останнім підписом і повідомленням у кінці відповідних списків. Підписом повідомлення в цій схемі будуть поточні стани списків повідомлень, підписів і публічних ключів OTS, що були використані. Тоді необхідно знайти:

$$O(T_{SIGN}) = O(T_{OTS\_GEN}) + O(T_{OTS\_SIGN}), \quad (6)$$

$$O(S_{SIGN}) = N * (O(S_{OTS\_msg}) + O(S_{OTS\_pk}) + O(S_{OTS\_SIGN})), \quad (7)$$

де *N* – кількість підписаних повідомлень, включно з поточним.

Для верифікації підпису потрібно пройти по всіх OTS-підписах із загального підпису та перевірити їхню дійсність, тобто:

$$O(T_{VRFY}) = N * O(T_{OTS\ VRFY}), \quad (8)$$

де *N* – кількість *OTS* підписів у наданому CBS підписі.

Як бачимо, розмір підпису та час його верифікації з кожним новим підписом зростає лінійно.

2.3. TBS (Tree-Based Signature)

Генерація приватного та публічного ключів повністю відповідають генерації приватного та публічного ключів відповідної схеми OTS. Для підписання нового повідомлення, треба спершу перевести його в HashSize-бітове значення, потім, відповідно до бітів повідомлення, спуститися по дереву від кореня до листка, паралельно створюючи нові вершини дерева, та підписуючи їхні публічні ключі, за потреби. За допомогою ж приватного ключа листка підписується безпосередньо саме повідомлення. Підписом повідомлення в цій схему будемо вважати списки підписів у вершинах на шляху від відповідного листка до кореня, а також відповідні публічні ключі з дочірніх вершин, що відповідають цим підписам. З огляду на це

$$O(T_{SIGN}) = HashSize * (O(T_{OTS\_GEN}) + O(T_{OTS\_GEN}) + O(T_{OTS\_SIGN})) + O(T_{OTS\_SIGN}), \quad (9)$$

$$O(S\_SIGN) = HashSize * (O(S\_(OTS\_pk)) + O(S\_(OTS\_pk)) + O(S\_(OTS\_SIGN))) + O(S\_(OTS\_SIGN)). \quad (10)$$

А для верифікації підпису потрібно пройти по всіх OTS-підписах із загального підпису та перевірити їхню дійсність, тобто:

$$O(T_{VRFY}) = HashSize * O(T_{OTS\_VRFY}). \quad (11)$$

Як бачимо, розмір підпису та час його верифікації з кожним новим підписом залишається сталим.

2.4. STBS (Sequential Tree-Based Signature)

Генерація приватного та публічного ключів повністю відповідають генерації приватного та публічного ключів відповідної схеми OTS. Для підписання нового повідомлення, необхідно взяти наступну в черзі обходу дерева в ширину вершину (можна легко відслідковувати за допомогою вказівників), створити дві дочірні вершини з новими парами ключів для них і за допомогою приватного ключа поточної вершини підписати повідомлення разом із публічними ключами двох її дочірніх вершин. Підписом повідомлення в цій схему будуть вважатися списки підписів у вершинах на шляху від відповідного листка до кореня, а також відповідні публічні ключі з дочірніх вершин і підписані раніше повідомлення, що відповідають цим підписам. Зважаючи на це,

$$O(T_{SIGN}) = O(T_{OTS\_GEN}) + O(T_{OTS\_GEN}) + O(T_{OTS\_SIGN}), \quad (12)$$

$$O(S_{SIGN}) = \log_2(N) * (O(S_{OTS\_msg}) + O(S_{OTS\_pk}) + O(S_{OTS\_pk}) + O(S_{OTS\_SIGN})), \quad (13)$$

де *N* – кількість підписаних повідомлень.

Для верифікації підпису треба пройти по всіх OTS підписах із загального підпису та перевірити їхню дійсність, тобто:

$$O(T_{VRFY}) = M * O(T_{OTS\_VRFY}), \quad (14)$$

де *M* – кількість OTS підписів у наданому STBS-підписі.

2.5. WSTBS (Wide Sequential Tree-Based Signature)

Ця схема ідейно повністю аналогічна попередній з тою відмінністю, що звичайні вершини дерева генерують не дві, а три вершини, тому:

$$O(T_{SIGN}) = O(T_{OTS\_GEN}) + O(T_{OTS\_GEN}) + O(T_{OTS\_GEN}) + O(T_{OTS\_SIGN}), \quad (15)$$

$$O(S_{SIGN}) = \log_2(N) * (O(S_{OTS\_pk}) + O(S_{OTS\_pk}) + O(S_{OTS\_pk}) + O(S_{OTS\_SIGN})) + O(S_{OTS\_msg}), \quad (16)$$

$$O(T_{VRFY}) = M * O(T_{OTS\_VRFY}), \quad (17)$$

де *N* – кількість підписаних повідомлень, а *M* – кількість OTS підписів у наданому WSTBS-підписі.

На перший погляд останні дві схеми переважають над схемою TBS як за часом, необхідним на підписання і верифікацію, так і за розмірами самого підпису. Натомість, як зазначається в книзі (Katz, & Lindell, 2014, p. 472–473), такий підхід теоретично дає можливість втілити схему без зберігання стану підписів. Отже, конкретизуємо: оскільки в основі всіх генерацій ключів лежать псевдовипадкові функції, за правильної реалізації, якщо знати корінне значення (сід) цих псевдовипадкових функцій, то будь-яку заздалегідь обрану пару публічного та приватного ключів одноразового підпису можна легко відтворити. Однак, ураховуючи, що всі значення, які підписуються цими приватними ключами, за фіксованого значення сіда повністю детерміновані, а від підписуваного повідомлення просто обирається, яка гілка дерева стане підписом, то в постійному підтриманні стану підписів немає потреби, оскільки, знаючи сід, можна легко відтворити весь підпис на ходу. І хоча надійність такого підходу значно спирається на правильний вибір псевдовипадкової функції, натомість зникає потреба в постійній підтримці та зберіганні стану підписів.

3. Опис реалізації основних функцій

У цьому підрозділі будуть розглянуті реалізації основних функцій GEN, SIGN та VRFY для кожного реалізованого протоколу.

3.1. Лампорт

Для генерації приватного ключа створюється масив розмірності 2xHashSize, що заповнюється гешами від випадково згенерованих значень. Після цього з нього формується публічний ключ, шляхом створення масиву такої самої розмірності, де кожне значення – геш від відповідного значення приватного ключа:

```
def gen() -> sk, pk:
    sk = [[SHA256(randomValue()) for j in 0..HashSize-1] for i in 0..1])
    pk = [[SHA256(sk[i][j]) for j in 0..HashSize-1] for i in 0..1])
    return sk, pk
```

Щоб підписати повідомлення, достатньо створити масив розмірності HashSize, який заповнити значеннями приватного ключа, відповідно до бітів підписуваного повідомлення:

```
def sign(sk, msg) -> sign:
    sign = [sk[((msg >> j) & 0b1)][j] for j in 0..HashSize-1])
    return sign
```

Для верифікації підпису достатньо переконатися, що геш-значення від елементів з підпису відповідають значенням публічного ключа, відповідно до бітів підписуваного повідомлення:

```
def vrfy(pk, sign, msg) -> bool:
    for j in 0.. HashSize-1:
        if SHA256(sign[j]) != pk[((msg >> j) & 0b1)][j]:
            return False
    return True
```

3.2. CBS (Chain-Based Signature)

Генерація пари ключів CBS повністю відповідає генерації пари ключів, обраної OTS, із тою відмінністю, що приватний ключ CBS насправді являє собою структуру, яка зберігає списками усі проміжні повідомлення, їх підписи, а також згенеровані приватні та публічні ключі від одноразових підписів:

```
def gen() -> sk, pk:
    otsk, otpk = OTS.gen()
    sk = {otsks: [otsk], otpks: [None], otsigns: [], msgs: []}
    pk = otpk
    return sk, pk
```

Під час підписання генерується нова пара ключів одноразового підпису, публічний ключ якої разом із повідомленням підписується поточним приватним ключем одноразового підпису. Після чого сформований підпис, разом із повідомленням і новоствореними приватним та публічним ключами додаються в кінці відповідних списків приватного ключа підпису CBS. У кінці формується підпис CBS, що складається зі списків одноразових публічних ключів, одноразових підписів і повідомлень із секретного ключа CBS:

```
def sign(sk, msg) -> sign:
    otsk, otpk = OTS.gen()
    otsign = OTS.sign(sk.otsks.last(), msg || otpk)
    sk.add(otsk, otpk, otsign, msg)
    sign = {otpks: sk.otpks, otsigns: sk.otsigns, msgs: sk.msgs}
    return sign
```

У процесі верифікації підпису відбувається перевірка всіх наданих одноразових підписів із підпису CBS. Для простоти перевірки, перший елемент зі списку одноразових публічних ключів замінюється на публічний ключ CBS:

```
def vrfy(pk, sign, msg) -> bool:
    if sign.msgs.last() != msg:
        return False
    sign.otpks[0] = pk
    for otpk, next_otpk, otsign, msg in sign:
        if not OTS.vrfy(otpk, otsign, msg || otpk):
            return False
    return True
```

3.3. TBS (Tree-Based Signature)

Генерація пари ключів TBS повністю відповідає генерації пари ключів обраної OTS, із тією відмінністю, що приватний ключ TBS насправді являє собою структуру, яка зберігає списком вершини дерева, кожна з яких або містить у собі певне повідомлення і його одноразовий підпис та відповідну пару одноразових ключів, або посилання на дві дочірні вершини, одноразовий підпис їхніх публічних ключів та відповідну пару одноразових ключів:

```
def gen() -> sk, pk:
    otsk, otpk = OTS.gen()
    sk = [{otsk: otsk, otpk: otpk, left: None, right: None, sign: None, msg: None}]
    pk = otpk
    return sk, pk
```

Під час підписання повідомлення відбувається спуск униз по дереву до листка, значенню якого відповідає підписуване повідомлення. Якщо на шляху у вершини не існує дочірніх вершин, – вони створюються разом із парами одноразових ключів підписів, і новостворені публічні ключі підписуються одноразовим приватним ключем поточної вершини. Після цього відбувається заглиблення в одну з дочірніх вершин, відповідно до біту вхідного повідомлення. Коли ж досягається листкова вершина, відбувається підписання самого повідомлення. Підпис при цьому має аналогічну структуру до підпису CBS, але будується з елементів на шляху до листка і також містить публічні ключі дочірніх вершин до всіх, що є на цьому шляху:

```
def sign(sk, msg) -> sign:
    cur = 0
    sign = {otpks: [], lotpks: [], rotpks: [], otsigns: [], msgs: []}
    for height in 0..HashSize-1:
        if sk[cur].sign == None:
            sk[cur].left = sk.size()
            lotsk, lotpk = OTS.gen()
            sk.add({otsk: lotsk, otpk: lotpk, left: None, right: None, sign: None, msg: None})

            sk[cur].right = sk.size()
            rotsk, rotpk = OTS.gen()
            sk.add({otsk: rotsk, otpk: rotpk, left: None, right: None, sign: None, msg: None})

            sk[cur].sign = OTS.sign(sk[cur].otsk, lotpk || rotpk)
            sk[cur].msg = None
        else:
            lotpk, rotpk = sk[sk[cur].left].otpk, sk[sk[cur].right].otpk

        sign.add(sk[cur].otpk, lotpk, rotpk, sk[cur].sign, sk[cur].msg)
        cur = ((msg >> height) & 0b1) ? sk[cur].right : sk[cur].left

    sk[cur].sign = OTS.sign(sk[cur].otsk, msg)
    sk[cur].msg = msg
    sign.add(sk[cur].pk, sk[cur].sign, sk[cur].msg)
    sign.otpks[0] = None
    return sign
```

Зважаючи на аналогічність структури підпису зі схемою CBS, код верифікації ідейно ідентичний:

```
def vrfy(pk, sign, msg) -> bool:
    if sign.msgs.last() != msg:
        return False
    sign.otpks[0] = pk
    for otpk, lotpk, rotpk, otsign, msg in pk:
        if not OTS.vrfy(otpk, otsign, msg || lotpk || rotpk):
            return False
    return True
```

3.4. STBS (Sequential Tree-Based Signature)

Генерація пари ключів STBS повністю дублює генерацію пари ключів TBS для обраної OTS із тією відмінністю, що кожна вершина дерева також має посилання на батьківську, а разом із самим деревом також зберігається вказівник на наступну вершину для підпису нового повідомлення:

```
def gen() -> sk, pk:
    otsk, otpk = OTS.gen()
    sk = {tree: [{otsk: otsk, otpk: otpk, prev: None, left: None, right: None, sign: None, msg: None}], head: 0}
    pk = otpk
    return sk, pk
```

Під час підписання повідомлення спершу береться наступна вершина в дереві. Для неї генеруються дві дочірні вершини, разом із парами одноразових публічних ключів кожна, і новостворені публічні ключі, разом із новим повідомленням, підписуються одноразовим приватним ключем поточної вершини. Після цього будується підпис STBS за принципом проходження від поточної вершини до кореня, у який також додаються публічні ключі дочірніх вершин до всіх, що є на цьому шляху:

```
def sign(sk, msg) -> sign:
    sign = {otpks: [], lotpks: [], rotpks: [], otsigns: [], msgs: []}
    cur = sk.head

    sk.tree[cur].left = sk.tree.size()
    lotsk, lotpk = OTS.gen()
    sk.tree.add({otsk: lotsk, otpk: lotpk, prev: cur, left: None, right: None, sign: None, msg: None})

    sk.tree[cur].right = sk.tree.size()
    rotsk, rotpk = OTS.gen()
    sk.tree.add({otsk: rotsk, otpk: rotpk, prev: cur, left: None, right: None, sign: None, msg: None})

    sk.tree[cur].sign = OTS.sign(sk.tree[cur].otsk, msg || lotpk || rotpk)
    sk.tree[cur].msg = msg

    while cur != None:
        lotsk, rotpk = sk[sk[cur].left].otpk, sk[sk[cur].right].otpk
        sign.addFromBegin(sk.tree[cur].pk, lotpk, rotpk, sk.tree[cur].sign, sk.tree[cur].msg)
        cur = sk.tree[cur].prev

    sk.head = sk.head + 1
    return sign
```

З огляду на аналогічність структури підпису зі схемою TBS, код верифікації ідейно ідентичний.

3.5. WSTBS (Wide Sequential Tree-Based Signature)

Генерація пари ключів WSTBS повністю дублює генерацію пари ключів STBS для обраної OTS із тією відмінністю, що кожна вершина дерева, окрім листків, також має посилання на листок дерева, яким уже підписується повідомлення:

```
def gen() -> sk, pk:
    otsk, otpk = OTS.gen()
    sk = {tree: [{otsk: otsk, otpk: otpk, prev: None, left: None, mid: None, right: None, sign: None, msg: None}], head: 0}
    pk = otpk
    return sk, pk
```

Під час підписання повідомлення спершу береться наступна вершина в дереві. Для неї генеруються три дочірні вершини разом із парами одноразових публічних ключів кожна, і новостворені публічні ключі підписуються одноразовим приватним ключем поточної вершини. Після цього будується підпис WSTBS за принципом проходження від поточної вершини до кореня:

```
def sign(sk, msg) -> sign:
    sign = {otpks: [], lotpks: [], motpks: [], rotpks: [], otsigns: [], msgs: []}
    cur = sk.head

    sk.tree[cur].left = sk.tree.size()
```

```
    lotsk, lotpk = OTS.gen()
    sk.tree.add({otsk: lotsk, otpk: lotpk, prev: cur, left: None, mid: None ,right: None, sign: None, msg: None})

    sk.tree[cur].mid = sk.tree.size()
    motsk, motpk = OTS.gen()
    sk.tree.add({otsk:motsk, otpk:motpk, prev:cur, left: None, mid: None ,right: None, sign: None, msg: None})

    sk.tree[cur].right = sk.tree.size()
    rotsk, rotpk = OTS.gen()
    sk.tree.add({otsk: rotsk, otpk: rotpk, prev: cur, left: None, mid: None ,right: None, sign: None, msg: None})

    sk.tree[cur].sign = OTS.sign(sk.tree[cur].otsk, lotpk || motpk || rotpk)
    sk.tree[cur].msg = None

    cur = sk.tree[cur].mid
    sk.tree[cur].sign = OTS.sign(sk.tree[cur].otsk, msg)
    sk.tree[cur].msg = msg

    while cur != None:
        lotsk, motpk, rotpk = sk[sk[cur].left].otpk, sk[sk[cur].mid].otpk, sk[sk[cur].right].otpk
        sign.addFromBegin(sk.tree[cur].pk, lotpk, motpk, rotpk, sk.tree[cur].sign, sk.tree[cur].msg)
        cur = sk.tree[cur].prev

    sk.head = sk.head mod 3 == 0 ? sk.head + 1 : sk.head + 2
    return sign
```

Зважаючи на аналогічність структури підпису зі схемою TBS, код верифікації ідейно ідентичний.

4. Заміри реалізованих цифрових підписів

Було виміряно час генерації ключів підпису, час підписання повідомлення та час верифікації у схемі Лампорта (табл. 2):

Таблиця 2

Час виконання методів схеми Лампорта

Кількість ітерацій	Середній час GEN, с	Середній час SIGN, с	Середній час VRFY, с
1	0.003451	0.000114	0.000749
100	0.003304	0.000166	0.000745
10000	0.003319	0.000172	0.000745
164187	0.003498	0.000175	0.000742

а також розмір згенерованих пар ключів та розмір підпису (табл. 3):

Таблиця 3

Розміри згенерованих пар ключів і підпису в схемі Лампорта

Кількість ітерацій	Середній розмір пари ключів, МБ	Середній розмір підпису, МБ
1	0.1956	0.0439
100	0.1839	0.0424
10000	0.1837	0.0424
164187	0.1837	0.0424

Розміри пар ключів і підпису повністю відповідають очікуванням, але ось час виконання функцій GEN, SIGN та VRFY перевищує їх орієнтовно в 4-5 разів. Проте в цьому випадку це зумовлено певними аспектами реалізації й обмеженнями мови.

Перейдімо тепер до багаторазових схем підписів на основі підписів Лампорта. Для замірів поведінки різних схем багаторазових підписів генерували повідомлення розміром 2560 Біт і послідовно їх підписували, а потім перевіряли відповідні підписи. Заміряли час підписування, час підписування з копіюванням, час верифікації, розмір підпису та загальне використання оперативної пам'яті. З огляду на різну асимптотичну складність функцій різних схем, кількість повідомлень, на яких проводили заміри, різна. Наприклад, CBS була перевірена на 1000 повідомленнях, TBS – на 10259, STBS – на 83550, а WSTBS – на 59052. Усі зібрані дані можна побачити на графіках, де CBS червоного кольору, TBS – помаранчевого, STBS – зеленого та WSTBS – синього (рис. 1).

Показники загального використання пам'яті переважно очікувані, оскільки з кожним підписом структура, що зберігає поточний стан підписів (список або дерево), просто лінійно зростає на однакову величину. Проте у випадку TBS заміри постійно показували константну величину. Це пояснюємо тим, що внаслідок псевдовипадковості геш-функцій більшість вершин на шляху від кореня до листочка буде зустрічатися нам уперше та генеруватись, тоді як раніше створені вершини, за невикористання, автоматично переносяться з оперативної пам'яті на жорсткий диск. Розглянемо інші заміри детальніше.

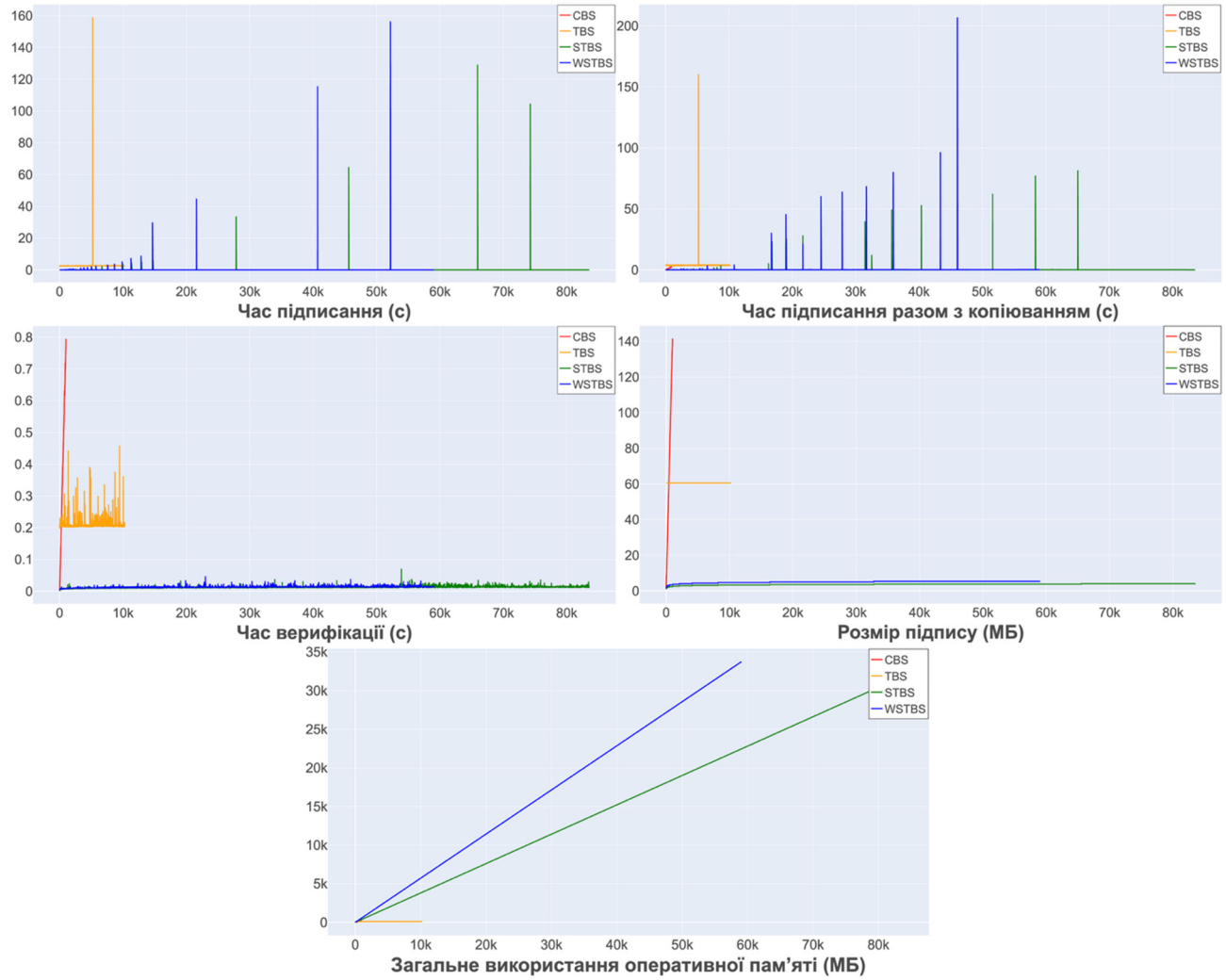

Рис. 1. Основні заміри

4.1. Час підписання

На обох графіках часу підписання видно послідовні сплески, що зростають. Проте єдина відмінність між графіками – це додаткові витрати часу на повне копіювання підпису, але на графіку з копіюванням сплесків помітно більше. Це легко пояснити внутрішніми процесами комп'ютера, на якому проводили заміри, що й викликали такий "шум". Конкретніше, більшість із них – це звільнення оперативної пам'яті під структуру, що зберігає підписи. Усунувши шум, матимемо (рис. 2):

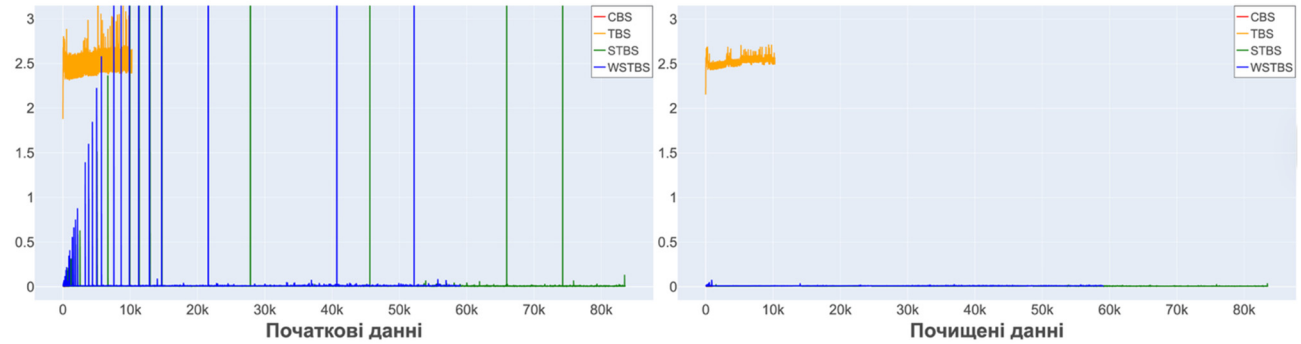

Рис. 2. Очищення даних часу підписання, с

Час, необхідний на підписання у схемі TBS, очікувано наближений до константного і практично відповідає теоретичним розрахункам, згідно з якими на підписання має йти 1.83 с. Якщо ж розглядати детальніше всі інші схеми та трохи усереднити дані, матимемо (рис. 3).

На графіках ще спостерігається "шум", але в іншому час теж наближений до константного та зберігає співвідношення відповідно до кількості генерацій ключів одноразового підпису за одного підписання.

4.2. Час верифікації

Розглянемо час верифікації (рис. 4).

У схемі CBS час верифікації росте лінійно, тоді як у схемі TBS час зберігається наближений до константного, що повністю відповідає асимптотичним оцінкам. Якщо ж виокремити схеми STBS та WSTBS, матимемо (рис. 5).

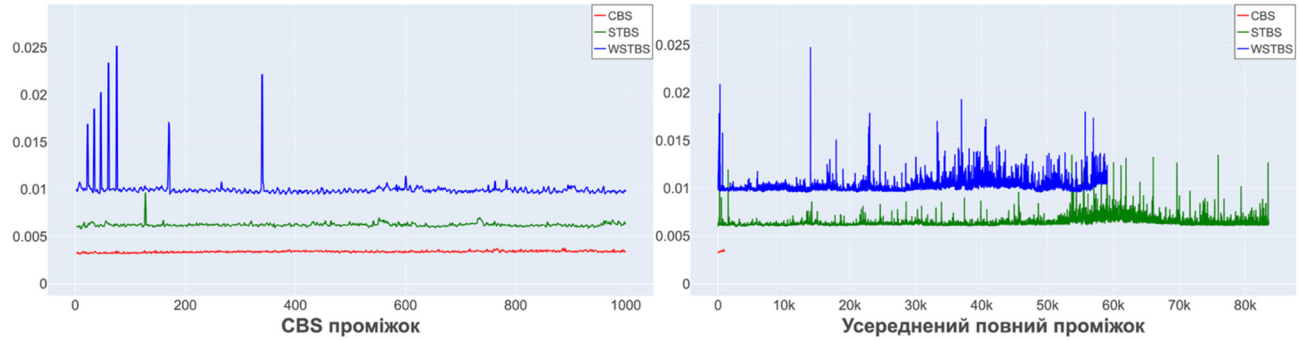

Рис. 3. Усереднені дані часу підписання, с

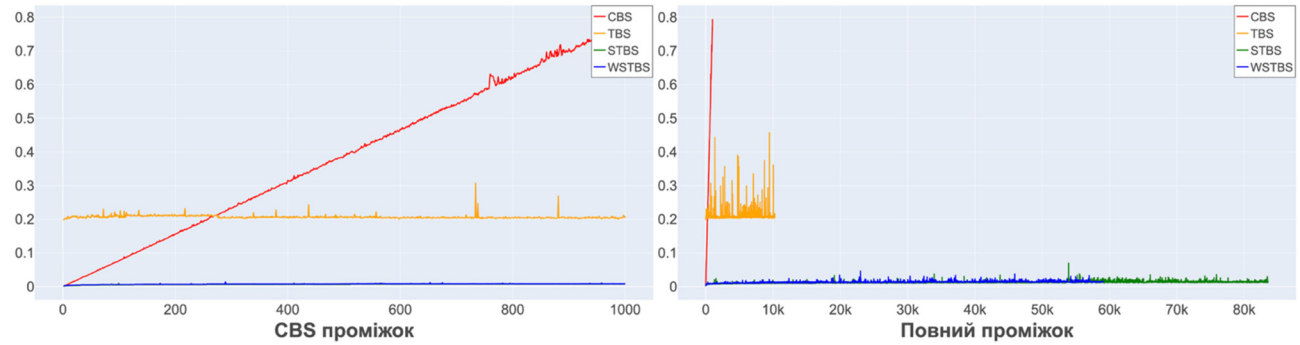

Рис. 4. Час верифікації, с

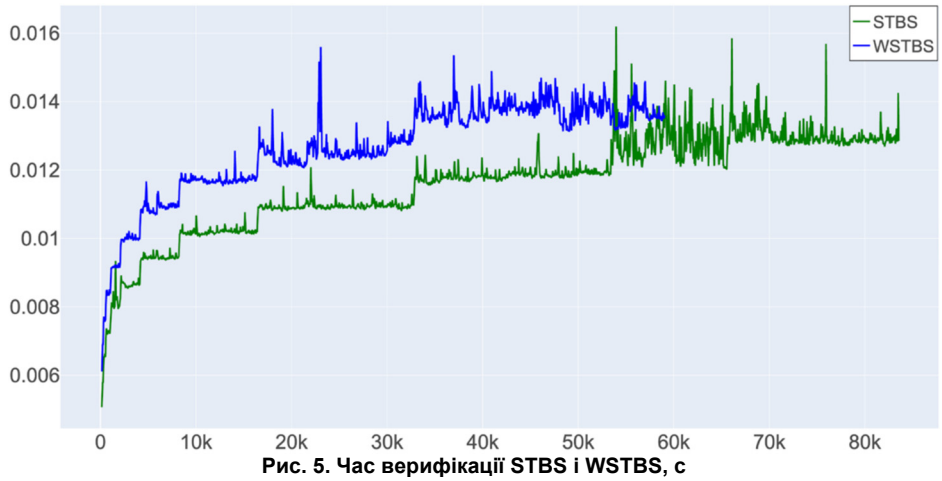

Рис. 5. Час верифікації STBS і WSTBS, с

Спостерігається логарифмічна асимптотика обох графіків з яскраво вираженими "сходинками" однакової висоти, що відповідають зростанню довжини підпису на одну ланку.

4.3. Розмір підпису

Розглянемо графіки розміру підпису (рис. 6).

Знову бачимо повну відповідність асимптотичним оцінкам. На останньому графіку також спостерігаємо невеликий "шум" та "сходинки" однакової висоти, що відповідають зростанню довжини підпису на одну ланку.

4.4. Загальний аналіз

З огляду на графіки можна побачити явну перевагу підписів на основі дерев над підписами на основі ланцюгів. Практично одразу спостерігаємо стрімке зростання як часу верифікації, так і розміру підпису, чого не відбувається в інших структур. Водночас час підписання не сильно відрізняється від інших схем, окрім TBS. Як результат маємо абсолютну неконкурентоспроможність.

Якщо ж розглядати суто деревоподібні схеми, то STBS та WSTBS явно переважають над схемою TBS на декілька порядків за всіма заміряними показниками, окрім пам'яті, необхідної для підтримання стану структури даних. Якщо ж останнє, навпаки, є важливим, то схема TBS за правильної реалізації є фаворитом.

ISSN 1728-3817

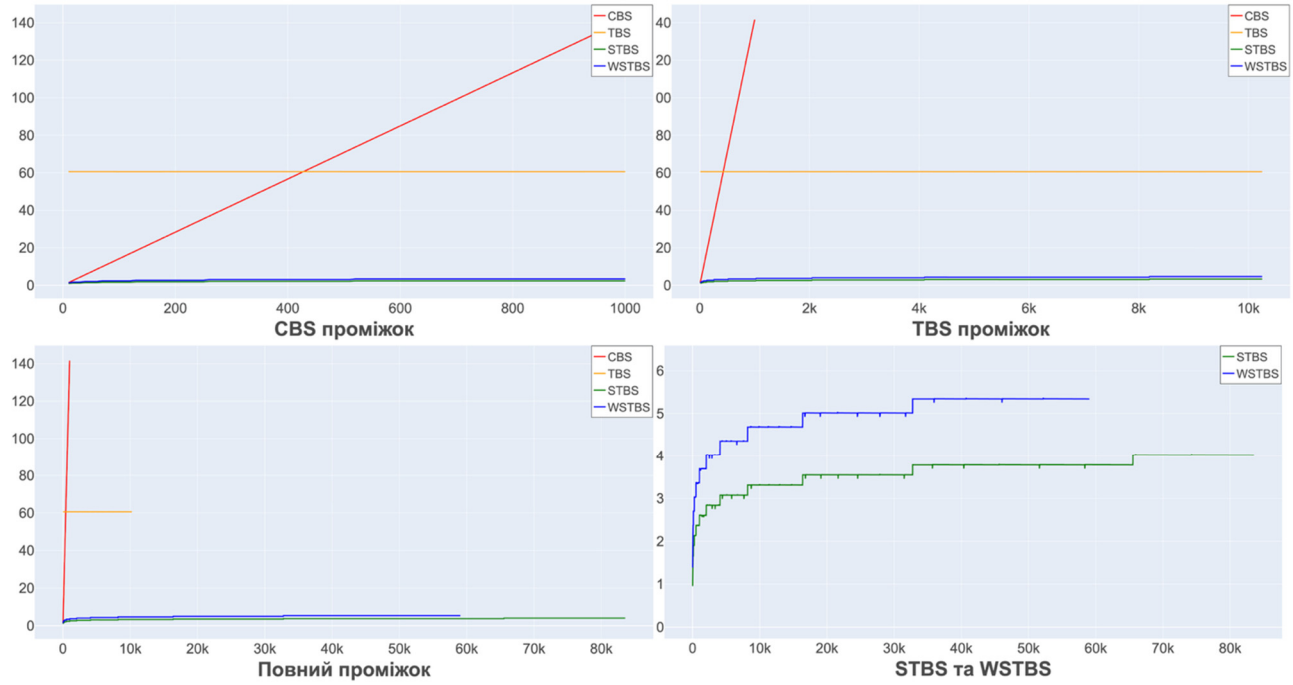

Рис. 6. Розмір підпису, МБ

Дискусія і висновки

У пропонованій роботі досліджені постквантові методи цифрових підписів, зокрема увагу було приділено схемі одноразового підпису Лампорта. Крім цього, були розглянуті та зіставлені й різні методи побудови схем багаторазового підпису на основі одноразових підписів.

Отримані результати показали, що найкращими з розглянутих методів побудови багаторазових підписів є схеми, що ґрунтуються на побудові бінарних дерев підпису. Проте вони мають як свої переваги, так і недоліки. Наприклад, якщо швидкодія за часом і малий розмір підпису не є основними цілями, то вигідніше скористатися схемою TBS, тоді як в іншому випадку доцільніше використовувати схеми STBS або WSTBS.

Варто зауважити, що в розглянутих схемах використовувалися лише бінарні дерева пошуку. З їхньою допомогою асимптотичну складність вдалося зменшити з лінійної до логарифмічної. Напрошується логічне припущення, що інші подібні структури збереження даних, зокрема й тернарні та N-арні дерева підписів, можуть дати хоча й асимптотично аналогічні оцінки, але порівняно кращі остаточні результати. З огляду на це наведені раніше структури мають бути розглянуті та досліджені в подальшому.

Джерела фінансування. Фінансування частково забезпечено Київським національним університетом імені Тараса Шевченка.

Список використаних джерел

Bleichenbacher, D., & Maurer, U. M. (1994). Directed Acyclic Graphs, One-way Functions and Digital Signatures. In Y. G. Desmedt (Eds.), *Lecture Notes in Computer Science: Vol. 839. Advances in Cryptology – CRYPTO '94* (pp. 75–82). Springer. https://doi.org/10.1007/3-540-48658-5_9

Boneh D., & Shoup, V. (2017). *A Graduate Course in Applied Cryptography*. https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_6.pdf

Hülsing, A. (2013). W-OTS+ – Shorter Signatures for Hash-Based Signature Schemes. In A. Youssef, A. Nitaj, & A. E. Hassanien, (Eds.), *Lecture Notes in Computer Science: Vol. 7918. Progress in Cryptology – AFRICACRYPT 2013* (pp. 173–188). Springer. https://doi.org/10.1007/978-3-642-38553-7_10

Katz, J., & Lindell, Y. (2014). *Introduction to Modern Cryptography, Second Edition*. Chapman and Hall/CRC. https://doi.org/10.1201/b17668

Lizama-Pérez, L. A. (2022). Digital signatures over HMAC entangled chains. *Engineering Science and Technology, an International Journal*, 32(101076). https://doi.org/10.1016/j.jestch.2021.11.002

Merkle, R. C. (1989). A Certified Digital Signature. In G. Brassard (Eds.), *Lecture Notes in Computer Science: Vol. 435. Advances in Cryptology – CRYPTO' 89 Proceedings* (pp. 218–238). Springer. https://doi.org/10.1007/0-387-34805-0_21

NIST (2015). *FIPS PUB 180-4: Secure Hash Standard (SHS)*. https://doi.org/10.6028/NIST.FIPS.180-4

Perrig, A. (2001). The BiBa one-time signature and broadcast authentication protocol. In P. Samarati (Eds.), *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security* (pp. 28–37). Association for Computing Machinery. https://doi.org/10.1145/501983.501988

Reyzin, L., & Reyzin, N. (2002). Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In L. Batten, & J. Seberry (Eds.), *Lecture Notes in Computer Science: Vol. 2384. Information Security and Privacy* (pp. 144–153). Springer. https://doi.org/10.1007/3-540-45450-0_11

References

Bleichenbacher, D., & Maurer, U. M. (1994). Directed Acyclic Graphs, One-way Functions and Digital Signatures. In Y. G. Desmedt (Eds.), *Lecture Notes in Computer Science: Vol. 839. Advances in Cryptology – CRYPTO '94* (pp. 75–82). Springer. https://doi.org/10.1007/3-540-48658-5_9

Boneh D., & Shoup, V. (2017). *A Graduate Course in Applied Cryptography*. https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_6.pdf

Hülsing, A. (2013). W-OTS+ – Shorter Signatures for Hash-Based Signature Schemes. In A. Youssef, A. Nitaj, & A. E. Hassanien, (Eds.), *Lecture Notes in Computer Science: Vol. 7918. Progress in Cryptology – AFRICACRYPT 2013* (pp. 173–188). Springer. https://doi.org/10.1007/978-3-642-38553-7_10

Katz, J., & Lindell, Y. (2014). *Introduction to Modern Cryptography* (Second Edition). Chapman and Hall/CRC. https://doi.org/10.1201/b17668

Lizama-Pérez, L. A. (2022). Digital signatures over HMAC entangled chains. *Engineering Science and Technology, an International Journal*, 32(101076). https://doi.org/10.1016/j.jestch.2021.11.002

Merkle, R. C. (1989). A Certified Digital Signature. In G. Brassard (Eds.), *Lecture Notes in Computer Science: Vol. 435. Advances in Cryptology – CRYPTO' 89 Proceedings* (pp. 218–238). Springer. https://doi.org/10.1007/0-387-34805-0_21

NIST (2015). *FIPS PUB 180-4: Secure Hash Standard (SHS)*. https://doi.org/10.6028/NIST.FIPS.180-4

Perrig, A. (2001). The BiBa one-time signature and broadcast authentication protocol. In P. Samarati (Eds.), *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security* (pp. 28–37). Association for Computing Machinery. https://doi.org/10.1145/501983.501988

Reyzin, L., & Reyzin, N. (2002). Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In L. Batten, & J. Seberry (Eds.), *Lecture Notes in Computer Science: Vol. 2384. Information Security and Privacy* (pp. 144–153). Springer. https://doi.org/10.1007/3-540-45450-0_11

Отримано редакцією журналу / Received: 25.06.24
Прорецензовано / Revised: 26.11.25
Схвалено до друку / Accepted: 16.04.26
Опубліковано / Published: 05.06.26

Oleksii BASHUK, PhD Student
ORCID ID: 0009-0002-6778-2943
e-mail: a.bashuk@knu.ua
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

COMPARATIVE ANALYSIS OF POST-QUANTUM DIGITAL SIGNATURE PROTOCOLS BASED ON HASH FUNCTIONS

In the era of active information technologies and device development, the emergence of portable and general-purpose quantum computers is only a matter of time. And although the benefit of their many times greater efficiency, compared to ordinary bit computers, will be obvious, at the same time, they will bring many vulnerabilities to modern algorithms of cryptography and cybersecurity. To prevent this from happening, society has already started researching post-quantum protocols to replace modern ones.

Digital signature protocols were no exception. Thus, one of the approaches was constructing post-quantum digital signature protocols based on hash functions. The first representatives were schemes of one-time signatures based on hash functions, which were protected from the capabilities of quantum computers but had the obvious disadvantage of one-time use. The most famous is the so-called Lamport one-time signature. However, this disadvantage of one-time use has been solved by proposing different schemes of multi-time digital signatures based on the creation and use of many one-time signatures. Thus, chain-based and tree-based schemes of multi-time signatures were proposed. This work is devoted to the implementation and comparison of various multi-time signature schemes.

During the research, theoretical estimates of memory usage and time required for the operations of various schemes were given. Further implementation of these schemes and final measurements showed the correspondence of theoretical estimates with empirical results and clearly demonstrated the advantages and disadvantages of different schemes. Thus, tree-based schemes showed the best performance, especially those where the tree is built sequentially. However, in certain cases, it makes sense to use a tree with full-depth construction.

Due to the relative novelty and small amount of research in this field, multi-time digital signatures have much room for further research, for example, considering N-ary trees and comparing them with binary ones.

Keywords: ***hash function, digital signatures, post-quantum protocols, one-time digital signatures, multi-time digital signatures, chain-based schemes, tree-based schemes.***

Автор заявляє про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.

The author declares no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; in the decision to publish the results.