УДК 004.056.5 **DOI:10.30837/rt.2025.3.222.01**

І.Д. ГОРБЕНКО, д-р техн. наук, О.Г. КАЧКО, канд. техн. наук, Я.А. ДЕРЕВ'ЯНКО

ОПТИМІЗАЦІЯ ОПЕРАЦІЙ ОБЧИСЛЕННЯ ТА ПЕРЕВІРКИ ЦИФРОВОГО ПІДПИСУ ДЛЯ СТАНДАРТУ FIPS 205. 2 ЧАСТИНА

Вступ

В [1, 2] показано методи та результати оптимізації базових операцій на основі алгоритмів SHA та SHAKE. Показано, що загальні алгоритми генерації ключів, вироблення та перевірки електронного підпису складаються з послідовних кроків, кожний з яких застосовує результат попереднього кроку, що виключає можливість застосування паралельних обчислень для цих алгоритмів.

Автори алгоритму Sphincs [3], який покладено в основу стандарту FIPS 205, запропонували оптимізовану версію алгоритму без застосування AVX команд (папка Optimized_Implementationta з застосуванням AVX (папка Additional_Implementations). Можливість застосування паралельного виконання за рахунок потоків не розглядається.

В даній роботі розглядаються засоби та результати оптимізації, в тому числі за рахунок паралельних потоків при реалізації окремих кроків алгоритмів. Оптимізація за рахунок застосування операцій AVX не розглядається.

Для виміру результатів оптимізації в якості базового застосовують реалізацію, надану в [3], папка Additional_Implementations. Для порівняння реалізації авторів Стандарту та авторів статті виконано на комп'ютері: процесор 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz, OS Windows 11, Version 24H2, Microsoft Visual Studio Community 2019, Version 16.11.21.

Для завдання результатів застосовують прискорення, тобто відношення часу виконання до оптимізації та часу виконання після оптимізації. Час виконання вимірюється в тактах процесора. Прискорення несуттєво залежить від криптостійкості, тому при викладанні результатів для окремих схем будуть наводитись результати тільки для максимального рівня криптостійкості, кінцеві результати для функцій генерування ключів, вироблення та перевірки електронного підпису будуть наведені для усіх рівнів криптостійкості.

Для зручності застосування імена функцій співпадають з іменами зі Стандарту [1].

Нагадаємо, що в якості відкритого ключа PK застосовують компоненти:

PK_SEED – seed для відкритого ключа;

PK_ROOT – корінь дерева.

В якості секретного ключа SK застосовують компоненти:

SK_SEED – seed для секретного ключа;

SK_PRF – компонент для генерації дайджеста повідомлення;

компоненти відкритого ключа.

В якості базових операцій застосовують функції PRF, Tl, H, F, оптимізація яких розглянуто в попередній статті.

Далі розглянуто засоби та результати оптимізації для окремих схем.

1. Оптимізація функцій для роботи з одноразовим підписом (Winternitz One-Time Signature, WOTS)

Функції даного модуля застосовують при генерації ключів, електронного підпису та для його перевірки. В якості параметрів цей модуль застосовую параметри:

n – довжина в байтах повідомлення для формування одноразового підпису, компонентів

секретного та відкритого ключів. В Стандарті *n* приймає значення 16, 24 та 32 в залежності від рівня криптостійкості (2, 3, 5 відповідно);

lgw – параметр Winternitz, довжина ланцюжка в бітах, визначає граничні значення цілих чисел, якими кодується повідомлення для підпису (lgw = 4, w = 16, граничні значення 0..15). Ці цілі числа (позначимо їх s[i]) і є секретними ключами для одноразового підпису. Кількість таких чисел в стандарті дорівнює len = 2n + 3. Вибір параметру lgw = 4 значно спрощує формування секретного ключа, кожному байту вхідного повідомлення відповідають 2 цілих числа: старші 4 біта – перше число, молодші – друге, саме такий спосіб обчислення застосовують далі

1.1. Генерація електронного підпису. Функція wots_sign Стандарту

Функцію застосовують при генерації електронного підпису. Підпис складається з len послідовностей, кожна послідовність завдовжки n байтів. Одному цілому числу відповідає один компонент електронного підпису, тобто підпис складається з len компонентів. Псевдокод для алгоритму обчислення компонентів підпису представлено на рис. 1.

```
Вхід.
M – повідомлення для підпису завдовжки n байтів;
SK_SEED – компонент SEED секретного ключа;
PK_SEED – компонент SEED відкритого ключа;
ADR – поточна інформація про дерево.
Вихід.
WOTS_SIG – масив з len байтових послідовностей завдовжки n байтів кожна.
1 Генерація масиву s цілих чисел завдовжки len чисел.
2 for i := 0 to len do                          Цикл генерації секретних ключів для підпису
3       TADRS :=ADR;                            Адресна структура для секретного ключа
        TADRS.type:= WOTS_PRF
        TADRS. KeyPairAddress = ADR. KeyPairAddress
        TADRS.ChainAddress := i
4       sk[i]:= PRF (PK_SEED || skADRS || SK_SEED)    Обчислення ключа
5 end for
6 for i := 0 to len do                          Цикл генерації компонентів підпису
7         TADRS :=ADR;                          Адресна структура для підпису
          TADRS.ChainAddress := i
8         tmp:= sk [i];
9         for j:= i to i + s[i] do              Ланцюжок гешів завдовжки s[i]
10              TADRS.HashAddress := j;
11              tmp  = F (PK_SEED || TADRS|| tmp)
12        end for
13        WOTS_SIG [i] = tmp
14 end for
```

Рис. 1. Функція wots_sign. Псевдокод

Алгоритм оптимізовано за рахунок паралельного виконання циклів, обмежених рядками 2 – 5 та 6 – 14, а також за рахунок більш ефективного виконання функцій PRF та F, що визначено в роботі [2].

1.2. Генерація відкритого ключа по електронному підпису. Функція wots_pkFromSig Стандарту

Функцію застосовують для перевірки електронного підпису, який складається з len рядків завдовжки n байтів. Результатом роботи функції є рядок байтів завдовжки n байтів. При генерації підпису обчислювався ланцюжок гешів s[i] разів (цикл, обмежений рядками

9 – 12). Для відновлення відкритого ключа обчислюється ланцюжок гешів w–s[i]–1 разів, що забезпечує загальну кількість разів, яка залишається постійною і дорівнює w–1.

Псевдокод функції представлено на рис. 2.

```
Вхід.
        WOTS_SIG –підпис, масив з len послідовностей завдовжки n байтів кожна.
        M – повідомлення для підпису завдовжки n байтів;
        PK_SEED – компонент SEED відкритого ключа;
        ADR – поточна інформація про дерево.
Вихід.
        pk – відкритий ключ, рядок байтів завдовжки n байтів.
        1 Генерація масиву s цілих чисел завдовжки len чисел.
        2 for i := 0 to len do                    Цикл відновлення елементів відкритого ключа
                  TADRS :=ADR;                          Адресна структура для відновлення
                  TADRS.ChainAddress := i
        3         tmp [i]:= WOTS_SIG [i];
        4         for j:= s[i] to w – s[i] – 1 do        Ланцюжок гешів завдовжки s[i]
        5                 TADRS.HashAddress := j;
        6                 tmp[i] := F (PK_SEED || TADRS|| tmp [i])
        7          end for
        8 end for
        9 TADRS :=ADR;                            Адресна структура для відновлення
        10 TADRS.type := WOTS_PK
        11 TADRS.KeyPairAddress = ADR. KeyPairAddress
        12     := Tl(PK.seed, wotspkADRS, tmp)          Загальний ключ
```

Рис. 2. Функція wots_pkFromSig. Псевдокод

Для оптимізації цикл, обмежений рядками 2 – 8, виконується паралельно, застосовується оптимізована функція Tl [2].

1.3. Обчислення відкритого ключа по секретному ключу. Функція wots_pkGen Стандарту

В попередніх двох функціях для кожного з len значень s[i] спочатку виконується ланцюжок обчислення гешів s[i] разів (першій алгоритм), а потім цей ланцюжок продовжується w–s[i]–1 разів. Тобто загальна довжина ланцюжка дорівнює w–1 незалежно від значення s[i]. Наступний алгоритм також обчислює відкритий ключ за допомогою ланцюжка завдовжки w–1. Таким чином, значення відкритого ключа може бути обчислено як за допомогою секретного ключа, що виконується саме цим алгоритмом і відновлено по підпису, що виконується при перевірці підпису. Псевдокод алгоритму наведено на рис. 3.

```
Вхід.
        SK_SEED – компонент SEED секретного ключа;
        PK_SEED – компонент SEED відкритого ключа;
        ADR – поточна інформація про дерево.
Вихід.
        pk – відкритий ключ, рядок байтів завдовжки n байтів.
1       for i:=0 to len do                        Цикл для секретних ключів
2                 TADRS :=ADR;                        Адресна структура для обчислення
                  TADRS.Type = WOTS_PRF
                  TADRS. KeyPairAddress = ADR. KeyPairAddress
                  TADRS. ChainAddress:= i
3                 sk [i] = PRF (PK_SEED ||  TADRS || SK_SEED)
4        end for
5        for i:=0 to len do                       Цикл для обчислення компонентів ключа
6                 TADRS :=ADR;                        Адресна структура для обчислення
                  TADRS. ChainAddress:= i
7                 for j:= 0 to w – 1 do                Ланцюжок гешів завдовжки w–1
8                         TADRS.HashAddress := j;
```

```
9                    tmp[i] := F (PK_SEED || TADRS|| sk[i])
10               end for
11           end for
12           TADRS :=ADR;                          Загальний ключ. Адресна структура
             TADRS.type = WOTS_PK
             TADRS.KeyPairAddress = ADRS.KeyPairAddress
13           pk:= Tl(PK_SEED, TADRS, sk, len)      Загальний ключ
```

Рис. 3. Функція wots_ pkGen. Псевдокод

Як і в попередніх алгоритмах цикли, обмежені рядками 1 – 4 та 5 – 11, виконуються паралельно. Застосовується оптимізована функція Tl [2]

1.4. Результати оптимізації функцій для роботи з одноразовим підписом

Результати оптимізації наведено в табл. 1.

Ім'я режиму включає алгоритм для обчислення гешу (SHAKE або SHA) та ознаку режиму оптимізації (пам'ять -s, час – t). Ім'я функції після оптимізації доповнюється символом _ в кінці, наприклад, wots_sign та wots_sign_. Для кожної функції визначається прискорення, наприклад S (wots_sign), яке визначається відношенням значень в попередніх двох рядках.

Таблиця 1

Одноразовий підпис

	SHAKE256s	SHA256s	SHAKE256f	SHA256f
wots_sign	898991	448186	730929	454118
wots_sign_	376800	190132	314923	243752
S (wots_sign)	2.39	2.36	2.32	1.86
wots_pkFromSig	928681	490984	799897	447234
wots_pkFromSig_	393050	197531	331191	223814
S(wots_pkFromSig)	2.36	2.49	2.42	2.00
wots_pkGen	1857485	1027216	1775210	909103
wots_pkGen_	588189	383740	506253	294956
S (wots_pkGen)	3.16	2.68	3.5	3.08

Прискорення практично для усіх функцій 2 та більше 2.

2. Оптимізація функцій для роботи з розширеним деревом Мерклі (eXtended Merkle Signature Scheme (XMSS))

Це розширення дозволяє застосовувати одноразовий підпис для підпису декількох повідомлень.

Дерево Мерклі – двійкове дерево, в якому на нижньому рівні в якості вузлів знаходяться листи, пара яких об'єднуються в один вузол для наступного рівня.

Параметри.

h' – висота дерева. Визначає геометричні параметри дереву, а саме кількість його листів leafs = $2^{h'}$, кожний лист дерева розглядається як відкритий ключ, для перевірки якого можна застосовувати один ключ, який є коренем дерева. Кожний ключ можна застосовувати для підпису одного повідомлення, тобто схему можна застосовувати для $2^{h'}$ повідомлень.

2.1. Генерування дерева Мерклі. Функція xmss_node

Відповідна функція рекурсивна, для нижнього рівня обчислює геш листа, наступні рівні виконують обчислення гешу для пари вузлів. Кінцевий результат – корінь дерева. Псевдокод функції наведено на рис. 4.

```
Вхід.
        SK_SEED – компонент SEED секретного ключа;
        PK_SEED – компонент SEED відкритого ключа;
        ADR – поточна інформація про дерево.
         – номер вузла;
         – номер рівня.
Вихід.
        PK_ROOT – корінь дерева.
        1 if z = 0 then                  Це лист. Обчислюємо відповідний pk
        2          ADR.type =  WOTS_HASH; ADR. KeyPairAddress = i
         3         wots_pkGen_(PK_ROOT, SK_SEED, ADR)
         4 else                          Обробка пари листів
         5          lnode :=xmss_node_(SK_SEED, PK_SEED, ADR, 2 * i, z – 1); Лівий
         6          rnode:=xmss_node_(SK_SEED, PK_SEED, ADR, 2 * i + 1, z – 1); Правий
         7          ADRS.Type :=TREE; ADRS.TreeHeight :=z; ADRS.TreeIndex :=i;
         8         Tmp [0]:= lnode; Tmp [1]:= rnode; Root:= H(PK_SEED, ADR, Tmp);
         9 end if
```

Рис. 4. Функція xmss_node. Псевдокод

Оптимізація функції виконується за рахунок застосування оптимізованих варіантів функції wots_pkGen, H.

2.2. Генерування підпису. Функція xmss_sign

Підпис – це WOTS підпис, який відповідає одному з шляхів по дереву Мерклі, починаючи з рівня 0 до кореня дерева. По номеру вузла idx визначається номер додаткового вузла, який відповідає парі. Для кожного рівня, крім самого верхнього містить додатково значення вузла пари, який дозволить перейти на наступний рівень. Додаткові вузли формують шлях аутентифікації. Довжина підпису дорівнює довжині відповідного WOTS підпису та довжині відповідного шляху аутентифікації PATH для рівнів 0, 1, h’ – 1, тобто (len * n) + (h’ * n). Псевдокод функції наведено на рис. 5.

```
Вхід.
        M – повідомлення для підпису, рядок байтів завдовжки n байтів
        SK_SEED – компонент SEED секретного ключа;
        PK_SEED – компонент SEED відкритого ключа;
ADR – поточна інформація про дерево.
         dx – номер вузла;
Вихід.
        XMSS_SIG – підпис, XMSS_SIG = WOTS_SIG || PATH
1 Визначення адреси для підпису і для шляху аутентифікації
        P_XMSS_SIG = XMSS_SIG; P_PATH = P_XMSS_SIG + len * n
2 Для усіх рівнів крім останнього визначення номера додаткового вузла і обчислення його
значення
        for j := 0 to h’ do
                k:= (ind / 2^j) mod 1              Номер суміжного вузла
                xmss_node_( P_PATH[j], SK_seed, k, j, PK_seed, PK_seed_n, adr);
        end for
3 Формування адресної структури
        ADR.Type = WOTS_HASH;    ADR. KeyPairAddress = idx
4 Обчислення WOTS_SIG
        wots_sign_( WOTS_SIG, M, SK_SEED, PK_SEED, ADR);
```

Рис. 5. Функція xmss_sign. Псевдокод

Оптимізація функції xmss_sign виконується за рахунок:

- попереднього визначення адрес початку підпису і шляху аутентифікації, що попереджує копіювання даних значних розмірів;
- застосування оптимізованих версій функцій xmss_node_, wots_sign_

2.3. Обчислення відкритого ключа по підпису. Функція xmss_pkFromSig

Фактично ця функція обчислює корінь дерева Мерклі за підписом.

Для обчислення відкритого ключа для нульового рівня застосовують функцію wots_pkFromSig, яка обчислює відкритий ключ, який відповідає WOTS_SIG. Цей ключ застосовують як лист для дерева Мерклі. Для обчислення кореня цього дерева застосовують шлях аутентифікації PATH з підпису (рис. 6).

Оптимізацію функції виконують за рахунок застосування виключення операції копіювання окремих полів WOTS та застосування оптимізованих функцій для обчислення листа дерева Мерклі (функція wots_pkFromSig_) та функції H.

```
Вхід.
        WOTS_SIG – підпис, рядок байтів завдовжки (len * n) +  (h' * n)
        M – повідомлення для якого підпис, рядок байтів завдовжки n байтів
        PK_SEED – компонент SEED відкритого ключа;
        ADR – поточна інформація про дерево.
         dx – номер вузла, для якого підпис;
Вихід.
        root – корень відповідного дерева
        1 Визначення адреси для підпису і для шляху аутентифікації
                P_XMSS_SIG = XMSS_SIG; P_PATH = P_XMSS_SIG + len * n
        2 Формування інформаційної структури для WOTS ключа
        ADR.Type:= WOTS_HASH; ADR. KeyPairAddress := idx
        3 Обчислення відкритого ключа WOTS  по підпису
        node:= wots_pkFromSig_( P_XMSS_SIG, M, PK_SEED, ADR);
        4 Формування інформаційної структури для  роботи з деревом
        ADR.Type = TREE; ADR. TreeIndex:= idx
        5 for k:= 0  to h' do     Для усіх рівнів дерева Мерклі
        6         ADR.TreeHeight:= k + 1        Коректування інформаційної структури
        7         if  idx / 2^k парне                Пошук пари та обчислення вузла
        8                 ADR. TreeIndex := ADR. TreeIndex/2
        9                 node:=H(PK_seed, ADR, node, P_PATH [k]);
        10      else
        11                ADR. TreeIndex := (ADR. TreeIndex – 1)/2
        12                node:=H(PK_seed, ADR, P_PATH [k], node);
        13      endif
        14 end for
        15 root:=node
```

Рис. 6. Функція xmss_pkFromSig. Псевдокод

2.4. Результати оптимізації функцій для розширеного дерева Мерклі

Функції для розширеного дерева Мерклі в явному вигляді в реалізації Sphincs не застосовують, тому в табл. 2 наведено тільки дані для реалізації авторів. Результати для розширеного дерева Мерклі наведено в табл. 2.

Таблиця 2

Розширене дерево Мерклі

	SHAKE256s	SHA256s	SHAKE256f	SHA256f
xmss_node_	53125445	68369655	3183055	1951479
xmss_sign_	100795718	68369655	5889182	3812993
xmss_pkFromSig_	258114	157727	239068	157114

3. Оптимізація функцій для роботи з гіпердеревом (Hypertree)

Гіпердерево є двійковим деревом, кожний вузел якого є розширеним деревом Мерклі. Висота цього дерева визначається параметром d (кількість рівнів дерев Мерклі). Так як висота дерева Мерклі дорівнює h', загальна висота гіпердерева дорівнює h = d * h'. На останньому рівні з номером d-1 знаходиться одне дерево Мерклі, корінь якого співпадає з PK_ROOT, тому для останнього рівня корінь дерева не обчислюють. На попередньому рівні знаходяться 2 дерева, на рівні з номером 0 знаходиться 2^d або $2^{h-h'}$ дерев Меркли

3.1. Генерація підпису для гіпердерева. Функція ht_sign

Псевдокод для генерації підпису для гіпердерева (рис. 7).

```
Вхід.
        M – повідомлення для підпису;
        SK_seed – seed для секретного ключа;
        PK_seed – seed для відкритого ключа;
        tree_idx – індекс дерева;
        leaf_idx –індекс листа.
Вихід.
    HT_SIG – підпис для гіпердереві
                                                        Рівень 0 гіпердерева
    1 ADRS:=0; ADRS.TreeAddress:= TreeIdx               Інформаційна структура
    2 HT_SIG:= xmss_sign (M, SK_sign, PK_seed, ADR, leaf_idx);  Підпис для вузла дерева
    3 root:= xmss_pkFromSig (HT_SIG, M, PK_seed, ADRS, leaf_idx)   Корінь дерева
    4 for j = 1 to d do                                 Цикл для решти рівнів
    5       leaf_idx := tree_idx mod 2^h'                   Індекс листа
    6       tree_idx:= tree_idx / 2^h'                      Індкс дерева
    7       ADRS.LayerAddress:= j                           Інформаційна
    8       ADRS.TreeAddress:= tree_idx                       структура
    9       SIG:=xmss_sign (root, SK_sign, PK_seed, ADRS, leaf_idx);  Підпис
    10      if j ≠ d – 1 then                       Корень для неостаннього рівня
    11              root:= xmss_pkFromSig (SIG, root, PK_seed, ADRS, leaf_idx)
    12      end if
    13      HT_SIG:= HT_SIG || SIG      Підпис для HT дерева
    14 end for
```

Рис. 7. Функція ht_sign. Псевдокод

Підпис для гіпердерева складається з підписів дерев Мерклі для кожного рівня і має довжину d * XMSS_SIG = d * ((len * n) + (h' * n)).

В якості повідомлення M для рівня 0 застосовують відкритий ключ, сформований для лісу дерев, в якості індексу дерева (tree_idx) та індексу листа (leaf_idx), застосовують значення, які визначаються дайджестом повідомлення, що підписується. Для решти рівнів ідентифікаторами листів є наступні h' бітів індексу дерева tree_idx, корінь дерева попереднього рівня застосовують для обчислення підпису для наступного рівня.

Оптимізація функції виконується за рахунок застосування оптимізованих версій функцій xmss_sign, xmss_pkFromSig

3.2. Перевірка підпису для гіпердерева. Функція ht_verify

Псевдокод для перевірки підпису для гіпердерева (рис. 8).

Для перевірки підпису спочатку виконується ініціалізація адресної структури та початкових значень адреси підпису її довжини (кроки 1 – 3), а далі обчислюється відкритий ключ по цифровому підпису (функція xmss_pkFromSig, крок 4). Виконується d кроків формування кореня та порівняння кореня для останнього рівня з компонентом PK_root відкритого ключа.

Для оптимізації функції ht_verify застосовують оптимізовані версії функції xmss_pkFromSig.

```
Вхід.
        M – повідомлення для підпису;
        HT_SIG – підпис для гіпердерева;
        PK_seed – seed для відкритого ключа;
        PK_root – root для відкритого ключа;
        tree_idx – індекс дерева;
        leaf_idx –індекс листа.
Вихід.
        Success (OK в разі успіху та ERROR в разі помилки)
1 Формування інформаційної структури        Рівень 0 гіпердерева
        ADRS:=0; ADRS.TreeAddress:= TreeIdx
2 xmss_sign_len:= (len * n) +  (h' * n)                    Розмір підпису
3 SIG:= SubST (HT_SIG, 0, xmss_sign_len)                   Підпис
4 pk:= xmss_pkFromSig (SIG, M, PK_seed, ADRS, leaf_idx)    pk
5 for j = 1 to d do                                 Решти рівнів
6       SIG:= SubST (HT_SIG, j * xmss_sign_len, xmss_sign_len)  Підпис
7       leaf_idx := tree_idx mod 2^h'               Індекс листа
8       tree_idx:= tree_idx / 2^h'
9       ADRS.LayerAddress:= j                       Інформаційна
10      ADRS.TreeAddress:= tree_idx                 структура
11      pk:= xmss_pkFromSig (SIG, pk, PK_seed, ADRS, leaf_idx)   pk
12 end for
13 if pk = PK_ROOT then success = OK else success = ERROR endif
```

Рис. 8. Функція ht_verify

3.3. Результати оптимізації функцій для гіпердерева

Результати оптимізації для функцій ht_sign, ht_verify наведено в табл. 3.

Таблиця 3

Результати оптимізації функцій ht_sign, ht_verify

	SHAKE256s	SHA256s	SHAKE256f	SHA256f
ht_sign	3625906354	2108292458	490611920	285012219
ht_sign_	900021510	551139349	114492525	70366362
S (ht_sign)	4.03	3.83	4.29	4.05
ht_verify	7777288	4134225	15733905	8917948
ht_verify_	2097061	1344683	3894410	2619106
S (ht_ verify)	3.71	3.07	4.04	3.40

Прискорення для усіх функцій більше ніж в три рази.

4. Ліс дерев (Forest of Random Subsets (FORS))

FORS – багаторазовий підпис для дайджеста повідомлення, який формується безпосередньо з повідомлення для підпису і, можливо, випадкового рядка, застосовує параметри:

a – висота дерева Мерклі, саме з цих дерев складається ліс;

$t = 2^a$ – кількість листів в дереві Мерклі;

k – кількість дерев Мерклі в лісі. Кожному дереву відповідає множина особистих ключів, кількість яких визначається кількістю листів (t) дерева Мерклі. Загальна довжина особистих ключів дорівнює t * n байтів. Ці рядки псевдовипадково генеруються з SK_SEED.

4.1. Генерація секретного ключа. Функція fors_skGen

Псевдокод функції fors_skGen наведено на рис. 9.

```
Вхід.
        SK_seed – seed для секретного ключа;
        PK_seed – seed для відкритого ключа;
        ADRS – інформаційна структура;
        idx – індекс ключа.
Вихід.
        sk – рядок байтів завдовжки n
1 Формування інформаційної структури
        SK_ADRS := ADRS
        SK_ADRS.Type = FORS_PRF
        SK_ADRS.KeyParaAddress := ADRS. KeyParaAddress
        SK_ADRS.TreeIndex:= idx
2 Обчислення sk
        Sk:= PRF (PK_seed, SK_seed, SK_ADRS)
```

Рис. 9. Функція fors_skGen. Псевдокод

Оптимізація функції виконується за рахунок застосування оптимізованого варіанту функції PRF.

Для обчислення секретного ключа для усіх вузлів можна застосовувати паралельне виконання, але в даному випадку навантаження на кожну гілку буде недостатнім для отримання виграшу від паралелізму.

4.2. Генерація дерева Мерклі для лісу. Функція fors_node

Псевдокод функції fors_node показан на рис. 10.

Як і попередні функції генерації дерева Мерклі функція рекурсивна. Якщо рівень дорівнює 0, то функція обчислює лист дереву – відповідний секретний ключ (fors_skGen).

Якщо рівень більше 0, то виконується рекурсивний виклик функції для попереднього рівня z – 1 і номера вузла 2 * i, що відповідає лівому вузлу (lnode) та номера вузла 2 * i + 1, що відповідає правому вузлу (rnode). Після завершення рекурсивного виклику будуть отримані значення для лівого та правого вузлів, для яких обчислюється об’єднане значення (функція H).

4.3. Генерація підпису. Функція fors_sign

Псевдокод функції fors_ sign наведено на рис. 11.

По заданому рядку байтів *md* функція формує масив цілих чисел *s* завдовжки *k* чисел, під кожне число виділяються наступні *a* бітів.

```
Вхід.
        SK_seed – seed для секретного ключа;
        PK_seed – seed для відкритого ключа;
        ADRS – інформаційна структура;
          – номер вузла;
          – номер рівня.
Вихід.
        root – корінь дерева, рядок байтів завдовжки n
1 if z = 0 then
2       sk:= fors_skGen (SK_seed, PK_Seed, ADRS, i)
3       ADRS.TreeHeight = 0; ADRS. TreeIndex = i
4       node:= F(PK_seed, ADRS, sk);
5 else
6       lnode := fors_node (SK_seed, PK_seed, ADRS, 2 * i, z – 1);
7       rnode := fors_node (SK_seed, PK_seed, ADRS, 2 * i + 1, z – 1);
8       ADRS.TreeHeight = z; ADRS.TreeIndex = i
9       node:= H (PK_seed, ADRS, lnode || rnode);
10 end if
11 return node
```

Рис. 10. Функція fors_node

```
Вхід.
        SK_seed – seed для секретного ключа;
        PK_seed – seed для відкритого ключа;
        ADRS – інформаційна структура;
        md – рядок байтів завдовжки ── байтів
Вихід.
        FORS_SIG – підпис, рядок байтів завдовжки k * (n + a * n)
1 Формування масиву s
2 for i = 0 to k do                                  Цикл для усіх чисел масиву s
3       L_ADRS := ADRS
4       pSK:= FORS_SIG + i * (1 + a) * n             Адреса для sk
5       pAuth:= pSK + n                              Адреса для Auth
6       pSK  := fors_skGen (SK_seed, PK_seed, i * t + s[j])
7       for j = 0 to a do                            for AUTH
8               r:= ─                                Номер суміжного вузла
9               pAuth []j] := fors_node ( SK_seed, PK_seed, ADRS,  r)
10      end for
12  end for
```

Рис. 11. Функція fors_sign. Псевдокод

Для формування підпису виконується цикл k раз, в якому для кожного числа з масиву *s* формується секретний ключ (функція fors_skGen).

Адреса для секретного ключа (pSK) визначається згідно з номером ітерації, за рахунок цього забезпечено можливість виконання циклу паралельно. Після генерації секретного ключа виконується цикл для визначення гешів відповідних пар для кожного з *a* рівнів.

Структура підпису для i-го значення з масиву s:

pSK

pAuth [0], pAuth [1], … pAuth [a – 1]

Оптимізація функції за рахунок паралельного виконання зовнішнього циклу і застосування оптимізованого варіанту для функцій fors_skGen, fors_node.

4.4. Генерація відкритого ключа з підпису. Функція fors_pkFromSig

Псевдокод функції наведено на рис. 12.

```
Вхід.
        FORS_SIG – підпис завдовжки k * (1 + a)*n
        PK_seed – seed для відкритого ключа;
        ADRS – інформаційна структура;
        md – рядок байтів завдовжки ── байтів
Вихід.
        FORS_PK – відкритий ключ згідно з підписом (рядок байтів завдовжки n)
1 Формування масиву s цілих чисел з рядка md.
2 for i = 0 to k do                  Цикл для усіх чисел масиву s
3       L_ADRS := ADRS; ind:= s [i]
4       pSK:= FORS + i * (1 + a) * n;   pAuth:= pSK + n Адреси sk,  Auth; sk:= pSK[i]
5       adr_l.TreeHeight:= 0; adr_l.TreeIndex:= i * (1 << A) + ind
6       pnode0:= F(PK_seed, adr_l, pSK)
7       for j = 0 to a do                    for AUTH
8               adr_l. TreeHeight:= j + 1; ti:= adr_l. TreeIndex
9               if ind / 2^j парне then
10                      adr_l. TreeIndex:= ti / 2
11                      p[0] := pnode0; p[1] := pAuth[j]
```

```
12              else
13                      adr_l. TreeIndex := (ti – 1) / 2;
14                      p[1] := pAuth[j];  p[0] := pnode0
15              end if
16              pnode0:= H (PK_seed, adr_l, p);
17      end for
18      root [i]:= pnode0
19 end for
20 tadr:= ADRS; tadr.type:= FORS_ROOTS; tadr.KeyPair:= adr.KeyPair.
21 FORS_PK:=Tl (PK_seed, tadr, root, k)
```

Рис. 12. Функція fors_pkFromSig. Псевдокод

Після формування масиву цілих *s* з *k* чисел (крок 1) виконується цикл для кожного з цих чисел (крок 2). Для кожної ітерації циклу встановлюється в якості початкової адресна структура, яка задається в списку параметрів (крок 3). Визначаються адреси для поточного секретного ключа та шляху аутентифікації, а також поточний секретний ключ (крок 4).

Згідно з секретним ключем обчислюють вузол для рівня 0 (pnode0) (кроки 5, 6), а далі згідно зі шляхом аутентифікації (кроки 7 – 18) – корінь для відповідного дерева Меркли (root [i]).

По значенням root [i] для *k* значень обчислюється загальний відкритий ключ (кроки 20, 21).

Саме відкритий ключ, сформований функцією fors_pkFromSig, застосовують в якості повідомлення для підпису в гіпердереві (дивись функції для гіпердерева).

Для оптимізації застосовують паралельне виконання зовнішнього циклу та оптимізовані внутрішні функції.

4.5. Результати оптимізації функцій для лісу

Функції fors_skGen та fors_node займають порівняльно менше часу, ніж решта функцій цієї групи, тому далі наведено результати тільки для останніх двох функцій.

Результати оптимізації для функцій fors_sign та fors_pkFromSig наведено в табл. 4

Таблиця 4

Результати оптимізації функцій fors_sign та fors_pkFromSig

	SHAKE256s	SHA256s	SHAKE256f	SHA256f
fors_sign	1721324445	1247174926	96257811	72954353
fors_sign__	380254469	263020840	18306279	13198027
S (fors_sign)	4.53	4.74	5.26	5.53
fors_pkFromSig	489274	534699	758640	603552
fors_pkFromSig__	134006	109762	132002	110104
S (fors_node)	3.65	4.87	5.75	5.48

Прискорення для функцій змінюється в діапазоні 3.65 – 5.75.

5. Комплексні функції

Розглядаються функції генерування ключів, вироблення та перевірки електронного підпису, які застосовують функції для усіх схем, що розглянуті вище.

5.1. Генерація ключів. Функція slh_keygen_internal

Псевдокод наведено на рис. 13.

Функція приймає в якості вхідних даних випадкові значення SK_seed, SK_prf та PK_seed і формує значення PK_root як корінь дерева Меркле висотою *d* (кроки 1 – 3).

Компоненти відкритого та секретного ключа записуються в рядки PK, SK відповідно (кроки 4 – 5).

Вхід.
SK_seed – seed для секретного ключа;
SK_prf – prf для секретного ключа;
PK_seed – seed для відкритого ключа.
Вихід.
SK – секретний ключ завдовжки 4 * n байтів;
PK – відкритий ключ завдовжки 2 * n байтів.
1. ADRS:= 0 Формування інформаційної структури
2. ADRS.LayerAddress:= d – 1
3. PK_root:= xmss_node (SK_seed, PK_seed, ADRS, 0, h')
4. PK:= PK_seed || PK_root
5. SK:= SK_seed || SK_prf || PK

Рис. 13. Функція slh_keygen_internal. Псевдокод

5.2. Вироблення електронного підпису. Функція slh_sign_internal

Електронний підпис складається:

- з псевдовипадкового масиву байтів R завдовжки n байтів, який залежить від повідомлення для підпису, секретного ключа, і, можливо, випадкового рядка байтів, наявність якого робить підпис неконстантним;
- підпису для лісу;
- підпису для гіпердерева.

Псевдокод для функції slh_sign_internal (рис. 14).

Вхід.
MSG – повідомлення для підпису;
MSG_len –довжина повідомлення;
SK –секретний ключ;
addrnd – визначає, чи застосовується константний підпис, = PK_SEED для константного підпису та випадковому рядку завдовжки n байтів, якщо підпис неконстантний.
Вихід.
SIG – підпис
SIG_len[1] –довжина підпису
1 R:= PRF (SK_prf || addrnd) Компонент підпису R
2 digest := HMsg(R, PK_seed, PK_root, MSG, MSG_len) Дайджест повідомлення
3 md, idxtree, idxleaf,:= DigestParse(digest)
4 ADR:=0 Ініціалізація ADR
5 ADR. TreeAddress:= idxtree
6 ADR.Type:= FORS_TREE
7 ADR. KeyPairAddress:= idxleaf
Підпис для обраного дерева лісу та його відкритий ключ
8 SIG_FORS := fors_sign (R, md, SK_seed, PK_seed, ADR)
9 PK_fors := fors_pkFromSig (SIG_FORS, md, PK_seed, ADR);
Підпис для гірердерева, для якого повідомлення дорівнює PK_fors
10 SIG_HT := ht_sign_(PK_fors, SK_seed, PK_seed, idxtree, idxleaf);
11 SIG:= R || SIG_FORS || SIG_HT

Рис. 14. Функція slh_sign_internal. Псевдокод

Після генерації R та дайджеста (digest) (кроки 1 – 3) інформація про дерево та його лист записується в інформаційну структуру ADR (кроки 4 – 7). Для заданого дерева лісу виробляють підпис і обчислюють відповідний відкритий ключ PK_fors (кроки 8, 9).

PK_fors далі застосовують в якості повідомлення для вироблення підпису гіпердерева.

Для оптимізації застосовують оптимізовані функції для виконання кожного кроку алгоритму.

[1] Параметри MSG_len, SIG_len визначаються, якщо програмний засіб для реалізації не передбачає автоматичне визначення довжини.

5.3. Перевірка електронного підпису. Функція slh_verify_internal

Псевдокод для функції slh_verify_internal задано на рис. 15.

Після перевірки довжини підпису (крок 2) виконується виділення окремих компонентів підпису, а саме R, підпису для лісу (SIG_FORS) та підпису для гіпердереву (SIG_HT).

```
Вхід.
        MSG – повідомлення для підпису;
        MSG_len – довжина повідомлення;
        PK – відкритий ключ;
        SIG – підпис;
        SIG_len–довжина підпису.
Вихід.
        Success – ознака успішності перевірки підпису. OK – успіх, ERROR – помилка
   1    Success:= OK
   2    If SIG_len = SIGSIZE then
   3            R:=Subst (SIG, 0, n)
   4            SIG_FORS:= Subst (SIG, n, k * (n + a * n))
   5            SIG_HT:= Subst (SIG, n + k * (n + a * n), d * (h' + len) * n)
                Дайджест повідомлення та його компоненти
   6            digest :=HMsg( R, PK_seed_, PK_root, M_, M_len, buf);        Дайджест
   7            md, idxtree, idxleaf,:= DigestParse(digest)
   8            ADR:=0                                          Ініціалізація ADR
   9            ADR. TreeAddress:= idxtree
   10           ADR.Type:= FORS_TREE
   11           ADR. KeyPairAddress:= idxleaf
                Формування повідомлення для гіпердерева
   12           PK_FORS := fors_pkFromSig (SIG_FORS, md, PK_seed, ADR) ;
                Перевірка підпису. Підпис повинен співпадати з PK_root
   13           Success:= ht_verify(PK_FORS, SIG_HT, PK_seed, idxtree, idxleaf,
                PK_root);
   14    end if
   15    return Success
```

Рис. 15. Функція slh_verify_internal. Псевдокод

Виконується формування дайджесту та його компонентів як для функції вироблення підпису (SIG_HT), кроки 3 – 5.

Формується дайджест повідомлення та його компоненти (кроки 6 – 8).

Поля інформаційної структури задаються як для вироблення підпису (кроки 8 – 11).

Генерується відкритий ключ за підписом для лісу (крок 12). Цей ключ застосовують як вхідне повідомлення для перевірки підпису для гіпердерева (крок 13) .

Для оптимізації застосовують оптимізовані функції для виконання кожного кроку алгоритму.

5.4. Результати оптимізації для комплексних функцій

Результати оптимізації представлені для усіх режимів застосування функцій, а саме криптостійкості 1, 3, 5 та внутрішніх функцій SHAKE, SHA, і містяться в табл. 5 – 7.

Таблиця 5

Результати оптимізації функцій для криптостійкості 1

	SHAKE128s	SHA128s	SHAKE128f	SHA128f
slh_keygen_internal	452180489	264897957	7431865	4739014
slh_keygen_internal__	109852714	69452559	1630782	1082316
S(Keygen)	4.12	3.81	4.56	4.38
slh_sign_internal	3495259228	1995710992	179101309	99506389
slh_sign_internal_	860563546	559322878	39058311	25313915
S(Sign)	4.06	3.57	4.59	3.93
slh_verify_internal	3745263	1909084	8968145	5458235
slh_verify_internal __	1141852	890416	3027383	1892022
S(Verify)	3.28	2.14	2.96	2.88

Таблиця 6

Результати оптимізації функцій для криптостійкості 3

	SHAKE192s	SHA192s	SHAKE192f	SHA192f
slh_keygen_internal	658694406	381176539	12869939	6998364
slh_keygen_internal __	162067955	98458804	2365196	1529277
S(Keygen)	4.06	3.87	5.44	4.58
slh_sign_internal	6086417491	3596133587	277799523	164115610
slh_sign_internal_	1556043730	1007007039	61471367	39621163
S(Sign)	3.91	3.57	4.52	4.14
slh_verify_internal	4695843	3053975	13664574	8326024
slh_verify_internal __	1626014	1160354	3943224	2619513
S(Verify)	2.89	2.63	3.47	3.18

Таблиця 7

Результати оптимізації функцій для криптостійкості 5

	SHAKE256s	SHA256s	SHAKE256f	SHA256f
slh_keygen_internal	444175185	256038156	26198709	19781809
slh_keygen_internal __	99349338	64456624	6241461	3985216
S(Keygen)	4.47	3.97	4.20	4.96
Sign	5582069024	3242691723	555418484	329285834
Sign_	1276065854	829180516	123821818	81778392
S(Sign)	4.37	3.91	4.49	4.02
slh_verify_internal	6803091	4435498	13412197	8411551
slh_verify_internal __	2177669	1547537	4083640	2750761
S(Verify)	3.12	2.87	3.28	3.06

Висновки

1 Для усіх функцій і усіх режимів отримано прискорення не менше ніж в два рази, для більшості функцій та режимів прискорення більше ніж в три рази.

2 Переважна більшість сучасних процесорів багатоядерна. Застосування паралельних обчислень за рахунок застосування багатоядерних процесорів суттєво збільшує продуктивність функцій для схем wots та fors, а також функцій, які їх застосовують.

3 Ефективність застосування AVX операцій для реалізації функцій алгоритму буде розглянуто в наступній частині.

Список літератури:

1. Stateless Hash-Based Digital Signature Standard, FIPS 205, 2024 [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf

2. Gorbenko I., Kachko O., & Derevianko Y. (2025). Optimization of digital signature calculation and verification operations for the FIPS 205 standard // Radiotekhnika. 2025. No 221. P. 7–13. https://doi.org/10.30837/rt.2025.2.221.01

3. NIST PQC. Round 3 Submissions. Алгоритм SPHINCS, Optimized_Implementation. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

Надійшла до редколегії 15.05.2025

Відомості про авторів:

Горбенко Іван Дмитрович – д-р техн. наук, професор, Харківський національний університет імені В.Н. Каразіна, професор кафедри кібербезпеки інформаційних систем, мереж і технологій, навчально-науковий інститут комп'ютерних наук та штучного інтелекту; АТ «Інститут інформаційних технологій», Голова наглядової ради; Україна; e-mail: i.d.gorbenko@karazin.ua; ORCID: https://orcid.org/0000-0003-4616-3449

Качко Олена Григорівна – канд. техн. наук, Харківський національний університет радіоелектроніки, професор кафедри програмної інженерії, факультет комп'ютерних наук; АТ «Інститут інформаційних технологій», член наглядової ради; Україна; e-mail: iit@iit.kharkov.ua, ORCID: https://orcid.org/0000-0001-9249-0497

Дерев'янко Ярослав Андрійович – Харківський національний університет імені В. Н. Каразіна, аспірант кафедри кібербезпеки інформаційних систем, мереж і технологій, навчально-науковий інститут комп'ютерних наук та штучного інтелекту, АТ «Інститут Інформаційних технологій», науковий співробітник-консультант; Україна; e-mail: yarik0009258@gmail.com; ORCID: https://orcid.org/0000-0002-3290-3373