УДК 004.056.5 **DOI:10.30837/rt.2025.2.221.01**

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

Вступ

Наразі суттєві зусилля на міжнародному та національному рівнях зосереджені на створенні практичних квантово-стійких механізмів цифрового підпису (ЦП). Проведено перший етап міжнародного конкурсу PQC [1], підсумком якого являється створення та стандартизація рекомендованих в якості міжнародних таких фіналістів 3 раунду конкурсу постквантових стандартів, але уже в якості федеральних стандартів США:

- FIPS 204, Стандарт ЦП на основі модульної решітки (алгоритм Crystals-Dilithium) [2];
- FIPS 205, Стандарт ЦП без стану на основі геш функції (алгоритм SPHINCS+) [3].

Вказані стандарти визначають схеми ЦП, які покликані протистояти майбутнім квантовим та класичним атакам квантових комп'ютерів, що загрожують безпеці існуючих стандартів. Оскільки ці алгоритми уже стандартизовано, то важливим завданням є дослідження їх побудови та практичної реалізації вимог до складових стандартів: побудови параметрів, генерування ключових пар, вироблення ЦП та їх верифікації тощо. Суттєво його вирішення залежить від покращення цих алгоритмів з точки зору складності (швидкодії), що може бути зведено до оптимізації базових операцій.

У даній статті розглядаються та пропонуються практичні удосконалення щодо оптимізації ЦП для алгоритму FIPS 205 на основі застосування паралельних обчислень, що зводяться в основному до оптимізації алгоритмів гешування shake256, sha256 та sha512. Важливість оптимізації обчислення геш значень пов'язана з тим, що гешування є основною операцією ЕП FIPS 205, а також використовується для FIPS 205.

1. Порівняння продуктивності базових алгоритмів

Аналіз показав, що згідно з класифікацією NIST алгоритми гешування забезпечують криптографічну стійкість алгоритму FIPS-204 на 2, 3, 5 рівнях та 1, 3, 5 рівнях для алгоритму FIPS 205.

Для порівняння стандартизованих алгоритмів ЕП зазвичай застосовують розміри ключових даних, насамперед відкритого ключа, розмір електронного підпису та час виконання основних операцій: генерації ключів, обчислення та перевірки електронного підпису. В табл. 1 та 2 наведені відповідні розміри параметрів для алгоритмів FIPS 204 та FIPS 205. В алгоритмі FIPS 205 передбачено два режими оптимізації[4]:

- за розміром цифрового підпису (режим позначено літерою s);
- часом обчислення цифрового підпису (режим позначено літерою f).

В табл. 2 наведені розміри цифрового підпису для обох режимів.

Таблиця 1

Алгоритм FIPS 204. Розміри ключів та цифрового підпису [2]

Параметри	Криптостійкість λ		
	2	3	5
Довжина секретного ключа	2560	4032	4896
Довжина відкритого ключа	1312	1952	2592
Довжина підпису	2420	3309	4627

Таблиця 2

Алгоритм FIPS 205. Розміри ключів та цифрового підпису [3]

Параметри	Криптостійкість λ					
	1		3		5	
	S	f	s	f	S	F
Довжина секретного ключа	64	64	96	96	128	128
Довжина відкритого ключа	32	32	48	48	64	64
Довжина підпису	7856	17088	16224	35664	29 792	49 856

Порівняння табл. 1 та 2 показує, що криптографічна стійкість для обох алгоритмів приблизно співпадає. При цьому розмір відкритого ключа для алгоритму FIPS 205 суттєво менший, що особливо важливо, оскільки саме відкриті ключі зберігаються в сертифікаті відкритого ключа чи передаються разом з підписаним повідомленням. Розмір ЦП для алгоритму FIPS 205 суттєво більший ніж для алгоритму FIPS 204, що також важливо враховувати, оскільки підпис передається разом з самим повідомленням.

Інформація щодо продуктивності алгоритмів FIPS 204 та FIPS 205, взята з технічної документації авторських реалізацій [5, 6] відповідно, наведена в табл. 3 (FIPS 204), 4 та 5 (FIPS 205). Продуктивність виміряється кількістю тактів процесора, що забезпечує мінімальну залежність від тактової частоти процесору.

Таблиця 3

Алгоритм FIPS 204. Продуктивність (Для AVX2 (Skylake)) [5]

Параметри	Криптостійкість λ		
	2	3	5
Медіана циклів для генерації	124031	256403	208050
Медіана циклів для підпису	259172	428587	538986
Середня кількість циклів для підпису	333013	529106	642192
Медіана циклів для перевірки	118412	179424	279936

Таблиця 4

Алгоритм FIPS 205. Продуктивність (3.1 GHz Intel Xeon E3-1220 CPU (Haswell), AVX2). Shake [6]

Параметри	Криптостійкість λ					
	1		3		5	
	s	f	s	s	f	s
Генерація	143 900 796	2 249 444	206 105 502	3 220 902	136 190 230	8 535 534
Підписання	1 102 470 520	56 933 788	1 910 461 606	89 875 552	1 650 717 926	176 951 378
Перевірка підпису	1 189 102	3 346 068	1 653 314	4 783 424	2 559 892	5 030 988

Таблиця 5

Алгоритм FIPS 205. Продуктивність (3.1 GHz Intel Xeon E3-1220 CPU (Haswell), AVX2). SHA2 [6]

Параметри	Криптостійкість λ					
	1		3		5	
	s	f	s	f	s	f
Генерація	84 964 790	1 334 220	125 310 788	1 928 970	80 943 202	5 067 546
Підписання	644 740 090	33 651 546	1 246 378 060	55 320 742	1 025 721 040	109 104 452
Перевірка підпису	861 478	2 150 290	1 444 030	3 492 210	1 986 974	3 559 052

Для алгоритму FIPS 204 кількість тактів для обчислення ЦП залежить від кількості повернень, тому в таблиці наведено мінімальне значення (Медіана циклів для підпису, немає повернень) та середнє значення (Середня кількість циклів для підпису).

Для алгоритму FIPS 205 кількість тактів для усіх операцій суттєво залежить від базового алгоритму для обчислення, тому результати наведені для обох варіантів:

- в табл. 4 – для варіанту застосування shake;
- в табл. 5 – для варіанту застосування sha2.

Не зважаючи на те, що результати отримані для різних процесорів, порівняння результатів показує, що за продуктивністю алгоритм FIPS 205 суттєво поступається алгоритму FIPS 204 для будь якої базової операції.

У наступному пункті розглядається оптимізація обчислення ЦП для алгоритму FIPS 205 за рахунок застосування паралельних обчислень.

2. Алгоритм обчислення ЦП для FIPS 205

У алгоритмі ЦП застосовуютьсяє наступні параметри:

n – довжина рядка байтів;

d – кількість рівнів розширених дерев Merkle (далі дерев Merkle).

h' – висота дерева Merkle;

h – загальна висота гіпер дерева (h = h' · d);

m – довжина дайджесту повідомлення, включає:

- інформацію для k листів завдовжки a бітів для кожного листа, всього k · a бітів;
- біти для завдання номеру найнижчого рівня гіпер дереву, який дорівнює h - h';
- біти для завдання номеру листа (максимальний номер h' - 1) .

len – загальна кількість листів в дереві Merkle;

k – кількість дерев в лісі.

a – Визначає загальну кількість листків t = 2^a для дерева FORS.

Вхідні дані в алгоритм :

msg – повідомлення довжини msg_len;

sk – секретний ключ. який включає:

- *SK_seed* (seed, випадковий рядок байтів завдовжки n),
- *SK_prf* (prf, випадковий рядок байтів завдовжки n)),
- *PK_seed* (seed, випадковий рядок байтів завдовжки n),
- *PK_root* (корінь геш дерева, рядок байтів завдовжки n)
- *opt_rand* – випадковий рядок байтів або PK_seed

Вихідні дані алгоритму:

sig – електронний підпис, який складається з:

- рандомізованого повідомлення *R;*
- підпису для лісу завдовжки $k \cdot (1 + a) \cdot n$ байтів;
- підпису для гіпер дерева завдовжки $(h + d \cdot len) \cdot n$ байтів.

Виконання алгоритму здійснюється у такій послідовності:

1. Рандомізація вхідного повідомлення (функція PRF_{msg} Стандарту).

Для рандомізації *msg* застосовують компонент секретного ключа *SK_prf, opt_rand*

$R = PRF_{msg}$ *(SK_prf, opt_rand, msg)*

2. Обчислення дайджесту для повідомлення (функція H_{msg} Стандарту) – рядок байтів завдовжки *m*

3 Обчислення по дайджесту *k* листів, номера рівня гіпердерева (номер дерева), та номера листа. Формування інформаційної структури, в яку записуються значення типу, яке відповідає *FORS_TREE*, значення номера дереву та номера листа в дереві (кроки 6-13 Стандарту)

4. Обчислення підпису для лісу SIG_{fors} (функція *fors_sign* Стандарту). Отримане значення записується як компонент електронного підпису. Цей підпис для дерева гешів включає значення вузлів і шляху аутентифікації

5. Обчислення відкритого ключа для гіпердерева за електронним підписом SIG_{fors} (функція *fors_pkFromSig* Стандарту). Ця функція за значеннями вузлів та шляху аутентифікації обчислює корінь відповідного дерева.

6. Обчислення підпису для гіпердерева (функція *ht_sign* Стандарту) Цей підпис додається як останній компонент електронного підпису.

Кожний наступний крок застосовує результат обчислення попереднього кроку, тому усі кроки необхідно виконувати послідовно. Необхідно розглянути можливість оптимізації кожного кроку окремо.

В наступному пункті розглянуто можливість паралельного виконання для базових операцій алгоритму FIPS 205.

3. Базові операції алгоритму FIPS 205

В якості базових операцій алгоритм застосовує операції, які в стандарті позначені як PRF_{msg}, H_{msg}, *PRF, Tl, H і F*. Їх виконання базується на обчисленні геш значень (алгоритми SHA2-256 та SHA2-512) або на алгоритмі SHAKE256.

Для алгоритму обчислення ЦП було визначено кількість застосовування базових операцій в залежності від параметрів алгоритмів (табл. 6).

Таблиця 6

Обчислення електронного підпису. Кількість викликів базових функцій

Функція	n = 16		n = 24		n= 32	
	s	f	s	f	s	F
PRF_{msg}	1	1	1	1	1	1
PRF	182784	8272	461312	17424	497664	36144
Tl	3584	176	3584	176	2048	272
H	60898	2230	282079	8566	362458	18136
F	1938674	94212	3019883	142683	2418193	290770
H_{msg}	1	1	1	1	1	1

Як видно з табл. 6, функції PRF_{msg}, H_{msg} викликаються тільки один раз, а функції PRF, Tl, H, F викликаються багаторазово, тому далі буде розглянуто оптимізацію саме цих функцій.

3.1. Алгоритм shake та його оптимізація

Для усіх наборів параметрів застосовують алгоритм shake256. Розмір блоку BLOCKSIZE (BS) для цього алгоритму дорівнює 136 байтів, внутрішній стан задається масивом s з 25 порцій даних завдовжки 64 біта.

Алгоритм shake є компонентом алгоритмів SHA3. Для завдання даних застосовується формат LITTLE-ENDIAN.

Усі операції запису в масив s фактично виконують операцію додавання по модулю 2 нового значення до поточного.

1. Ініціалізація (init).
   s[i] = 0 (I = 0, 1, 2, …, 25)
2. Накопичення для повних блоків (*absorb*)
   Поки розмір вхідних даних перевищує *BS* то
   Запис наступної порції в масив *s ^= por[i]*
   Перемішування для масиву *s*:
   s = KeccakF1600_StatePermute(s);
   Коректування масиву s з урахуванням неповної порції
   s ^= por
3. Кінцева обробка *(finalize).*
   Запис значення 0x1f безпосередньо після останнього біту неповної порції
   s[i] ^= 0x1F
   Запис в *s[16]* значення 2^{63}
   s[16] ^= 2^{63}
   Перемішування для масиву *s*

В оригінальній реалізації алгоритму враховується можливість завдання довжини вхідних даних в бітах, що суттєво ускладнює процес накопичення даних в масиві *s*, в даному оптимізованому алгоритмі довжина даних задається тільки в байтах, тому усі обчислення, пов’язані

з накопиченням даних суттєво спрощуються. Розмір рядка байтів для результату може бути більше, ніж розмір блоку, в даному алгоритмі це неможливо, розмір блоку завжди перевищує розмір рядка результату.

Для оптимізації обчислення для алгоритму *shake* також враховується, що для функцій *PRF, H, F* розмір вхідних даних не перевищує розміру блоку BS (Максимальна довжина вхідного блоку дорівнює $3 \cdot n + 32$, що дорівнює 128 *для* $n = 32$), тому не треба виконувати кроки 1, 2 алгоритму, достатньо в масив s записати вхідні дані, а для решти елементів масиву задати значення *0. Значення 0x1f записується в s[inlen / 8*].

В разі, якщо є більше одного блоку даних (функція **Tl**), при перемішуванні даних для кожного кроку дані спочатку копіюються в локальний масив, а потім виконується відновлення масиву. Якщо заздалегідь буде відома кількість блоків, є можливим виконання цих копіювань тільки для останньої ітерації.

В табл. 7 наведено параметри для оптимізації обчислень базових функцій для SHAKE256.

Таблиця 7

Параметри для оптимізації обчислень базових функцій для SHAKE256

Функція	Загальний розмір даних	Кількість повних блоків	Розмір неповного блоку
PRF	n + 32 + n	0	n + 32 + n
Tl	n + 32 + l · n (l = LEN, K)	(n + 32 + LEN · n)/BS (n + 32 + K · n)/BS	(n + 32 + LEN · n) mod BS (n + 32 + K · n) mod BS
H	n + 32 + 2 · n	0	(n + 32 + 2 · n)
F	n + 32 + n	0	n + 32 + n

Для функцій PRF та F довжина вхідного повідомлення співпадає, функція для обробки вхідного повідомлення теж співпадає, тому далі функція F не розглядається.

3.2. Алгоритм функцій sha256, sha512 та його оптимізація

Алгоритми є компонентами SHA-2.

1. Вхідні дані розглядаються як масив *32 (64)* бітних даних відповідно для *sha256, sha512*. Вхідні дані записуються з урахуванням формату *BIG-ENDIAN*. Перетворення для даних може виконуватись паралельно

2. Вхідні дані діляться на блоки розміром *64 (128)* байтів для *sha256, sha512* відповідно. Блоки обробляються послідовно, результат обробки попереднього блоку застосовують як вхідні дані для наступного. При обробці блоку паралелізм також недопустимий

3. Для решти даних розміром менше *64* байтів виконується доповнення до *64* байтів та обробка останнього блоку або двох блоків.

Таблиця 8

Параметри для оптимізації обчислень базових функцій для SHA256-SHA512

Функція	Загальний розмір даних	Кількість повних блоків	Розмір неповного блоку
PRF	BS + 22+ n	0	22+ n
Tl	BS + 22 + l · n (l = LEN, K)	(22 + LEN · n)/BS (22 + K · n)/BS	(22 + LEN · n) mod BS (22 + K · n) mod BS
H	BS + 22 + 2 · n	(BS + 22 + 2 · n)/BS	(BS + 22 + 2 · n) mod BS
F	BS + 22 + l · n (l = LEN, K)	(22 + LEN · n)/BS (22 + K · n)/BS	(22 + LEN · n) mod BS (22 + K · n) mod BS

Розмір блоку *(BS)* дорівнює *64* байти для *SHA256* та *128* байтів для *SHA512*.

В усіх функціях значення гешу для першого блоку визначається значенням *PK_seed* і може бути обчислене заздалегідь.

Кількість додаткових блоків та розмір неповного блоку залежать тільки від параметрів. Вони можуть бути попередньо обчислені, що спростить обробку доповнення останнього блоку.

4. Результати оптимізації

Результати оптимізації представлено в табл. 9 (SHAKE) та 10 (SHA256, SHA512). Перший рядок – кількість тактів до оптимізації, другий – після оптимізації. Для кожного значення, приведеного в таблицях, обчислюється мінімальне значення з 256 експериментів, що забезпечує мінімальну залежність від випадкових факторів переключення потоків. При порівнянні значень має сенс враховувати тільки дві старші цифри з врахуванням округлення.

Функція Tl викликається для вхідних даних розміром $LEN \cdot n$ та K · n, тому кількість тактів визначається для обох випадків.

Таблиця 9

Результати оптимізації (SHAKE)

Функція	128s	128f	192s	192f	256s	256f
PRFTime	1476 1277	1616 1471	1634 1462	1637 1480	1694 1454	1643 1448
TlTime (LEN)	(LEN = 35) 7200 6076	(LEN = 35) 8034 6950	(LEN = 51) 14951 13361	(LEN = 51) 14831 13310	(LEN = 67) 24297 22170	(LEN = 67) 24284 22184
TlTime (K)	(K = 14) 4720 3940	(K = 33) 8046 6952	(K = 17) 6820 5637	(K = 33) 10994 9504	(K = 22) 9567 8162	(K = 35) 13677 12132
Htime	1515 1310	1618 1450	1617 1466	1643 1474	1663 1480	1657 1449

Таблиця 10

Результати оптимізації (SHA)

Функція	128s	128f	192s	192f	256s	256f
PRFTime	1834 834	1810 890	1833 835	1917 839	1870 837	1872 866
TlTime (LEN)	(LEN = 35) 9977 7992	(LEN = 35) 9946 7973	(LEN = 51) 11942 10088	(LEN = 51) 11994 9780	(LEN = 67) 19821 17264	(LEN = 67) 19810 17420
TlTime (K)	(K = 14) 4674 3343	(K=33) 9060 7211	(K = 17) 6272 4203	(K = 33) 9151 7004	(K = 22) 8298 6152	(K = 35) 12252 9977
Htime	1827 825	1802 821	2356 1106	2320 1079	2343 1159	2341 1103

В табл. 11 наведено коефіцієнти прискорення (покращення), які обчислюються як відношення кількості тактів для стандартної оптимізованої версії до кількості тактів з урахуванням оптимізації авторів. В першому рядку задається коефіцієнт прискорення для алгоритму SHAKE, другий – для алгоритмів SHA.

Таблиця 11

Прискорення для базових операцій за використання різних функцій для алгоритму FIPS 205

Функція	128s	128f	192s	192f	256s	256f
PRFTime	1.16 2.2	1.1 2.03	1.12 2.2	1.11 2.28	1.17 2.23	1.13 2.16
TlTime (LEN)	LEN = 35 1.18 1.25	(LEN = 35) 1.16 1.25	(LEN = 51) 1.12 1.18	(LEN = 51) 1.11 1.23	(LEN = 67) 1.1 1.15	(LEN = 67) 1.1 1.14
TlTime (K)	(K = 14) 1.2 1.4	(K = 33) 1.26	(K = 17) 1.21 1.49	(K = 33) 1.16 1.31	(K = 22) 1.17 1.35	(K = 35) 1.13 1.23
Htime	1.16 2.2	1.12 2.2	1.10 2.13	1.11 2.15	1.12 2.02	1.14 2.12

Висновки

З отриманих у роботі результатів оптимізації можна зробити наступні висновки:

1 Операції для SHAKE при застосовуванні стандартних функцій, виконуються трохи скоріше, ніж відповідні операції для **SHA**, але останні можуть бути оптимізовані за рахунок постійного першого блоку для більшості базових операцій. Після оптимізації операції для SHA виконуються скоріше, ніж відповідні операції для SHAKE.

2 Застосування оптимізації забезпечує мінімальне прискорення для усіх операцій і всіх параметрів – 10 %.

3 Базова операція PRF, еквівалентна їй по списку параметрів та операцій функція F та функція H, згідно з табл. 6, викликаються найбільшу кількість разів. Прискорення для цих функцій в разі застосування SHA є не меншим ніж вдвічі.

4 Алгоритм FIPS 205 в якості базового формату застосовує формат *BIG-ENDIAN*, аналогічний формат застосовують функції групи SHA2. Для застосування однакового формату в подальших розширеннях алгоритму рекомендується застосовувати саме алгоритми SHA256 та SHA512.

Список літератури:

1. National Institute of Standards and Technology. (2017, January) // Post-Quantum Cryptography [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography

2. Module-Lattice-Based Digital Signature Standard, FIPS 204, 2024 [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf

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

4. D. Moody, R. Perlner, A. Regenscheid, A. Robinson, and D. Cooper. Transition to Post-Quantum Cryptography Standards // NIST Internal Report 8547 (Initial Public Draft) [Online], November 2024. Available: https://nvlpubs.nist.gov/nistpubs/ir/2024/NIST.IR.8547.ipd.pdf

5. National Institute of Standards and Technology. (2020, October) // Post-Quantum Cryptography. Round 3 Submissions. CRYSTALS-DILITHIUM. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

6. J.-P. Aumasson, D. J. Bernstein, W. Beullens, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, and B. Westerbaan //"SPHINCS+: Submission to the NIST Post-Quantum Project, v3.1 [Online], June 2022. Available: https://sphincs.org/data/sphincs+-r3.1-specification.pdf

Надійшла до редколегії 06.03.2025

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