УДК 004.056.5 **DOI:10.30837/rt.2022.3.210.01**

І.Д. ГОРБЕНКО, д-р техн. наук, О.Г. КАЧКО, канд. техн. наук,
М.В. ЄСІНА, канд. техн. наук, В.А. ПОНОМАР, канд. техн. наук

ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА АЛГОРИТМІВ ІНКАПСУЛЯЦІЇ КЛЮЧІВ CRYSTALS-KYBER ТА СКЕЛЯ (ДСТУ 8961-2019)

Вступ

Алгоритми інкапсуляції ключів (Key-establishment Algorithms, KEA) формують загальний секрет – ключ для симетричного алгоритму шифрування.

В роботі розглянуто два KEA алгоритмів, які застосовують алгебраїчні решітки: один з фіналістів 3-го раунду Crystals-Kyber [1] (далі Kyber) та алгоритм Скеля (ДСТУ 8961-2019) [2] (далі Скеля).

Алгоритм Kyber спочатку виконує несиметричне шифрування повідомлення завдовжки 32 байти, а потім виконується формування загального секрету.

Алгоритм Скеля виконує ті ж дії, але для несиметричного шифрування застосовує повідомлення будь-якої довжини, яка не перевищує максимально можливої. Ось чому останній алгоритм можна застосовувати не тільки як KEA алгоритм, а і як алгоритм несиметричного шифрування.

Згідно з NIST Security level алгоритм Kyber забезпечує криптографічну стійкість 1, 3 та 5 рівнів, а алгоритм Скеля забезпечує криптографічну стійкість 3, 5 та 7 рівнів. Криптографічна стійкість, яка забезпечується, для обох алгоритмів визначається набором параметрів.

Далі розглянуто деталі реалізації кожного з алгоритмів, виконано порівняння алгоритмів генерації ключів, інкапсуляції та декапсуляції для алгоритмів Kyber та Скеля з боку довжин ключових даних, і результату інкапсуляції та обчислювальної складності обох алгоритмів.

Алгоритми застосовують випадкові послідовності. Позначимо функцію для генерації випадкової послідовності gen_rand, яка приймає в якості вхідних даних довжину в байтах і повертає випадковий рядок.

1. Алгоритм Kyber

1.1. Параметри

Набір параметрів залежить від криптографічної стійкості. Його наведено в табл. 1.

Для кожного набору параметрів є два типи алгоритмів:

- алгоритм за замовчуванням;
- алгоритм, позначений як KYBER_90S.

У залежності від типу алгоритму застосовують різні алгоритми для гешування та генерації псевдовипадкових послідовностей (потокове шифрування).

Таблиця 1

Параметри для алгоритму Kyber

Позначення	Призначення	Значення
λ	Рівень криптостійкості	1, 3, 5
n	Степінь полінома x^n+1	256
q	Модуль	3329
k	Розмір вектора, компонентами якого є поліном	2 *для* $\lambda = 1$ 3 *для* $\lambda = 3$ 4 *для* $\lambda = 5$
η_1	Визначає граничні значення для коефіцієнтів поліномів s, e	3 *для* $\lambda = 1$ 2 *для* $\lambda = 3$ 2 *для* $\lambda = 5$
η_2	Визначає граничні значення для коефіцієнтів поліномів e_1, e_2	2 для $\lambda = 1$ 2 для $\lambda = 3$ 2 для $\lambda = 5$
(du, dv)	du – кількість бітів для коефіцієнту полінома при перетворенні вектора поліномів в рядок байтів dv – кількість бітів для елементу при перетворенні полінома в рядок байтів	10,4 для $\lambda = 1$ $10,4$ для $\lambda = 3$ $11,5$ для $\lambda = 5$
m_len	Довжина повідомлення для шифрування	32

Різниця між цими алгоритмами наведена у табл. 2.

Таблиця 2

Алгоритм Kyber – різниця між алгоритмом
за замовченням та алгоритмом KYBER_90S

Тип алгоритму	Гешування	Потокове шифрування
За замовченням	H – SHA3-256 G – SHA3-512	XOF – SHAKE-128 KDF – SHAKE-256 PRF – SHAKE-256
KYBER_90S	H – SHA-256 G – SHA-512	XOF – AES-256 в режимі CTR KDF – SHAKE-256 PRF – AES-256 в режимі CTR

Довжини гешів: H_len=32, G_len=64 байтів.

Як показали експериментальні дослідження для обох типів алгоритмів, обчислювальна складність алгоритму за замовчуванням менше, тому в подальшому передбачається застосування саме цього режиму.

1.2. Алгоритми Parse

Застосовуються при генерації поліномів з невід'ємними коефіцієнтами в інтервалі [0, q–1] (Parse^q) та при генерації малих поліномів з цілими коефіцієнтами, діапазон для яких задається параметрами η_1, η_2. (Parse$^\eta$). У тестових прикладах автори застосовують LITTLE ENDIAN представлення даних в пам'яті. Нумерація бітів в байті починається справа, тобто біт 0 – самий правий біт байту.

1.2.1. Алгоритм Parse^q для матриці

Алгоритм застосовує квадратну матрицю розміром k рядків. Елементом матриці є поліном степені n, коефіцієнти полінома – невід'ємні значення, задані в діапазоні [0, q–1]. Для інкапсуляції та декапсуляції застосовують транспоновану матрицю.

Для генерації псевдовипадкової послідовності застосовують XOF алгоритм. Ініціалізація виконується для кожного елементу матриці, тобто для кожного полінома. Для ініціалізації генератора застосовують *seed*, номер рядка і номер колонки матриці. У разі відновлення транспонованої матриці номер рядка і номер колонки міняються місцями. Довжина псевдовипадкового рядка buf_len$\approx\lceil log_2 q * n/(bytelen * p)\rceil$, де $bytelen$ – кількість бітів в байті

(зазвичай, 8), p – імовірність, що число завдовжки $log_2 q$ біт буде менше, ніж q. Для Kyber q=3329, n=256, $log_2 q$=12, $bytelen$=8, $p=q/2^{12}$=0.8127, buf_len≈473 (байтів). В тестових прикладах послідовність генерується блоками, розмір блоку для XOF алгоритму дорівнює 64 байти, тому buf_len=512 байтів.

Для кожного з n коефіцієнтів полінома в рядку псевдовипадкових байтів функція виділяє наступні $log_2 q$ бітів, перевіряє, що відповідне значення менше, ніж q та приймає це значення в якості наступного коефіцієнту полінома матриці. Якщо значення не менше, ніж q, тоді таке значення ігнорується.

Якщо вичерпано усю псевдовипадкову послідовність, генерується додаткова послідовність завдовжки один блок.

П р и к л а д . Хай буфер з псевдовипадковими даними починається з байтів:

0x4d 0x50 0xe8.

В пам'яті ці дані розташовані наступним чином:

Номер байту 2 1 0.

Значення байту 0xe8 0x50 0x4d.

Молодшим 12 бітам відповідає значення 0x04d=77<q – приймається, перший коефіцієнт полінома дорівнює 77.

Наступним 12 бітам відповідає значення 0xe85=7434≥q – не приймається.

1.2.2. Алгоритми Parse$^{\eta}$ для малих поліномів

Для генерації псевдовипадкових даних застосовують алгоритм PRF, для ініціалізації – відповідний *seed* та значення *nonce*, яке збільшується від 0 на 1 при кожному наступному застосуванні. Довжина псевдовипадкового рядка buf_len=2*η*n/$bytelen$. Для Kyber n=256, η=2 або 3, $bytelen$=8, $buf\_len = \begin{cases} 128 \text{ для } \eta = 2 \\ 192 \text{ для } \eta = 3 \end{cases}$.

Отриманий рядок байтів ділиться на бітові послідовності завдовжки η біт та обчислюється сума бітів кожної послідовності. Отримані суми розглядаються парами. Позначимо перший елемент пари, який відповідає молодшим бітам як a, другий елемент пари – b, тоді результат дорівнює $a–b$.

П р и к л а д 1 (η = 2) . Хай перші 4 байти байтової послідовності дорівнюють 0xE0 0x33 0x56 0x15

0xE0	0x33	0x56	0x15
1110 0000	0011 0011	0101 0110	0001 0101
Порції	Порції	Порції	Порції
11 10 00 00	00 11 00 11	01 01 01 10	00 01 01 01
Суми	Суми	Суми	Суми
a0 = 0 + 0 = 0;	a2 = 1 + 1 = 2;	a4 = 0 + 1 = 1	a6 = 1 + 0 = 1
b0 = 0 + 0 = 0;	b2 = 0 + 0 = 0;	b4 = 1 + 0 = 1	b6 = 1 + 0 = 1
a1 = 0 + 1 = 1;	a3 = 1 + 1 = 2;	a5=1 + 0 = 1	a7 = 1 + 0 = 1
b1 = 1 + 1 = 2;	b3 = 0 + 0 = 0;	b5= 1 + 0 = 1	b7 = 0 + 0 = 0
r[0] = a0 – b0 = 0	r[2] = a2 – b2=2	r[4] = a4 – b4 = 0	r[6] = a6 – b6 = 0
r[1] = a1 – b1=–1	r[3] = a3 – b3=2	r[5] = a5 – b5 = 0	r[7] = a7 – b7 = 0

П р и к л а д 2 (η = 3) . Хай перші 3 байти байтової послідовності дорівнюють 0xE0 0x33 0x56, тобто

Номер байту	2		1		0	
Значення байту	0x56		0x33		0xE0	
	01010110		0011 0011		1110 0000	
Порції						
B3	A3	B2	A2	B1	A1	B0	A0
010	101	100	011	001	111	100	000
Суми							

A0 = 0 B0 = 1 r[0] = -1
A1 = 3 B1 = 1 r[1] = 2
A2 = 2 B2 = 1 r[2] = 1
A3 = 2 B3 = 1 r[3] = 1

1.3. Алгоритми пакування та розпакування

Алгоритми пакування $pack^{q}$, $pack^{du}$, $pack^{dv}$ застосовують для перетворення поліномів та векторів у відповідні рядки байтів. Алгоритм $pack^{q}$ застосовують для запису компонентів секретного та відкритого ключа, алгоритми $pack^{du}$, $pack^{dv}$ – для пакування вектора ($pack^{du}$) та полінома ($pack^{dv}$), які є результатом шифрування.

Алгоритми розпакування ($unpack^{q}$, $unpack^{du}$, $unpack^{dv}$) виконують зворотні операції.

1.3.1. Алгоритми $pack^{q}$, $unpack^{q}$ для полінома (pol_pack^{q}, pol_unpack^{q})

1. Усі коефіцієнти полінома приводяться до інтервалу [0.. q–1].

2. Пара коефіцієнтів записується в наступні $\log_2 q$=12 бітів, перший коефіцієнт пари займає молодші, наступний – старші 12 бітів.

У результаті пакування для полінома отримуємо рядок байтів завдовжки poly_len=n*$\log_2 q$/bits_in_byte=384 (байт).

Функція pol_unpack^{q} є зворотньою до функції pol_pack^{q}.

1.3.2. Алгоритми $pack^{du}$ та $pack^{dv}$ для полінома (pol_pack^{du}, pol_unpack^{du}, pol_pack^{dv}, pol_unpack^{dv})

1. Усі коефіцієнти полінома приводяться до інтервалу [0.. q–1].

2. Усі коефіцієнти полінома обчислюються за формулою

$$coef_i := (coef_i * 2^d + q/2)/q,$$

де $d=du$ для pol_pack^{du} та $d=dv$ для pol_pack^{du}.

3. Кожний коефіцієнт записується в пам'ять в наступні d біт, починаючи з молодших байтів.

Алгоритм $pack^{dv}$ застосовують для перетворення поліномів, а $pack^{du}$ – для перетворення елементів вектора.

Функції pol_unpack^{du}, pol_unpack^{dv} є зворотними до функцій pol_pack^{du}, pol_pack^{dv}.

У результаті пакування для полінома отримуємо рядок байтів завдовжки $poly\_len^{dv}=n*dv/bits\_in\_byte=\begin{cases}128 & dv=4\\160 & dv=5\end{cases}$ (байт).

1.3.3. Алгоритми $pack^{q}$, $pack^{du}$ для вектора (vec_pack^{q}, vec_unpack^{q}, vec_pack^{du}, vec_unpack^{du})

У разі застосовування векторів поліномів кожний поліном, починаючи з початку, пакується окремо (алгоритм pol_pack^{q} або pol_pack^{du}) та записується як наступний блок пам'яті. В результаті отримуємо блок пам’яті завдовжки

$$vec\_len = k * poly\_len.$$

Функції vec_unpack^{q}, vec_unpack^{du} є зворотними до функцій vec_pack^{q}, vec_pack^{du}.

Для параметрів алгоритму

$$vec\_len^{q}=\begin{cases}768 & k=2\\1152 & k=3\\1536 & k=4\end{cases} \qquad vec\_len^{du}=\begin{cases}640 & k=2, du=10\\960 & k=3, du=10\\1408 & k=4\ du=11\end{cases}.$$

1.4. Перетворення повідомлення в поліном та зворотне перетворення. Функції pol_from_msg, pol_to_msg

В алгоритмі повідомлення завдовжки 32 байті (256 бітів) і поліном містить 256 коефіцієнтів, тобто один біт повідомлення відповідає одному коефіцієнту полінома та навпаки.

Біт 0 відповідає нульовому коефіцієнту. Біт 1 – коефіцієнту (q+1)/2.

1.5. Генерація ключів

Вихідними даними функції генерації ключів є рядки байтів *pk*, *sk* завдовжки pk_len, sk_len для відкритого та секретного ключів відповідно.

1. Формується випадковий рядок байтів завдовжки seed_len

buf:=gen_rand (seed_len).

2. Обчислюється геш (функція G) для цього рядка завдовжки 2 seed_len, перша половина обчисленого значення застосовується як псевдовипадковий рядок public_seed для створення матриці, а друга половина – в якості псевдовипадкового рядка (noise_seed) для генерації малих поліномів.

3. Генерується квадратна матриця (A), яка має *k* рядків та колонок (алгоритм $Parse^{q}$).

4. Генеруються вектори vec_sk, vec_e завдовжки *k* елементів кожний. Елементом вектора є поліном, коефіцієнти якого цілі числа (алгоритм $Parse^{\eta 1}$).

5. Обчислюється вектор vec_pk=A*vec_sk+vec_e.

6. *sk*, sk_len_internal:=vec_pack^{q}(vec_sk, *k*).

7. *pk*, pk_len:=vec_pack^{q}(vec_pk, k)||public_seed.

8. Генерація випадкового рядка *seed* завдовжки seed_len.

9. *sk*, sk_len:=*sk*||*pk*||*H*(*pk*)||*seed*

Довжина відкритого ключа: pk_len=vec_len+seed_len,

$$pk\_len = \begin{cases} 800 & k = 2 \\ 1184 & k = 3 \\ 1568 & k = 4 \end{cases} \text{ (байтів).}$$

Довжина секретного ключа

sk_len_internal=vec_len (без відкритого ключа),

sk_len=vec_len+pk_len+H_len+seed_len,

$$sk\_len = \begin{cases} 1632 & k = 2 \\ 2400 & k = 3 \\ 3168 & k = 4 \end{cases} \text{ (байтів).}$$

1.6. Інкапсуляція

Виконується несиметричне шифрування обраного повідомлення завдовжки 32 байти за допомогою відкритого ключа отримувача (функція encrypt).

Виконується обчислення загального секрету визначеної довжини.

В х і д .

pk – відкритий ключ отримувача, масив байтів завдовжки pk_len.

В и х і д .

ct – інкапсульований ключ (масив байтів завдовжки crypt_len).

ss – загальний секрет (масив байтів завдовжки).

1. Генерація випадкового рядка *seed* завдовжки seed_len та обчислення його гешу за допомогою функції *H* (саме це значення буде застосовуватись в якості повідомлення для зашифрування):

m:=*H*(*seed*, seed_len).

2. Обчислення гешу від відкритого ключа за допомогою функції *H* buf2:=*H*(*pk*, pk_len).

3. Обчислення гешу за допомогою функції *G*

kr:=*G*(*m*||buf2, 2*seed_len).

4. Обчислення *kr*1 (ліва половина) та *kr*2 (права половина), такі, що

kr:=*kr*1||*kr*2 (*kr*1, *kr*2 завдовжки seed_len).

5. Шифрування *m* за допомогою відкритого ключа отримувача (*pk*) та псевдовипадкового даного *kr*2

ct:=encrypt (*m*, *pk*, *kr*2).

6. Обчислення гешу для *ct* за допомогою функції *H*

$$kr2:=H(ct, \text{crypt\_bytes}).$$

7. Обчислення *ss* за допомогою функції KDF

$$ss:=\text{KDF}(kr1||kr2, 2*\text{seed\_len}).$$

1.6.1. Функція шифрування (encrypt)

В х і д .
m – повідомлення для шифрування, рядок байтів завдовжки seed_len;
pk – відкритий ключ отримувача, рядок байтів завдовжки pk_len;
seed – псевдовипадкове дане завдовжки seed_len.
В и х і д .
ct – результат шифрування повідомлення *m* (рядок байтів завдовжки crypto_len).
1. Ініціалізація *nonce*:=0.
2. Розпаковка відкритого ключа: vec_pk, seed_pk:=vec_unpack^{q}(*pk*).
3. Обчислення полінома pol_p:=pol_from_msg(*m*).
4. Відновлення матриці згідно з seed_pk в транспонованому вигляді (алгоритм Parse^{q})

$$At:=\text{Parse}^{q}(\text{seed\_pk}).$$

5. Генерація вектора *s* згідно з *seed* та поточним значенням nonce (алгоритм Parse$^{\eta 1}$)

$$\text{vec\_s, nonce:= Parse}^{\eta 1}(seed, nonce).$$

6. Генерація вектора vec_e згідно з *seed* та поточним значенням nonce (алгоритм Parse$^{\eta 2}$)

$$\text{vec\_e, } nonce:=\text{Parse}^{\eta 2}(seed, nonce).$$

7. Генерація полінома *ep* згідно з *seed* та поточним значенням nonce (функція Parse$^{\eta 2}$)

$$\text{pol\_ep, } nonce:= \text{Parse}^{\eta 2}(seed, nonce).$$

8. Обчислення:

$$\text{vec\_u}:=At*\text{vec\_s}+\text{vec\_e}; \text{ pol\_v}:=\text{vec\_pk}*\text{vec\_s}+\text{pol\_ep}+\text{pol\_p}.$$

9. Формування інкапсульованого ключа *cc*

$$cc:=\text{vec\_pack}^{du}(\text{vec\_u})||\text{pol\_pack}^{dv}(\text{pol\_v}).$$

Довжина інкапсульованого ключа:

$$\text{cc\_len}=n*(k*du+dv)/\text{bits\_in\_byte};$$

$$\text{cc\_}len=\begin{cases} 768, & k=2, du=10, dv=4 \\ 1088, & k=3, du=10, dv=4 \\ 1568, & k=4, du=11, dv=5 \end{cases}.$$

1.7. Декапсуляція

Генерація загального секрету по зашифрованому тексту *cc* та секретному ключу *sk*.
В х і д .
cc – зашифрований текст;
sk – секретний ключ.
В и х і д .
ss – загальний секрет завдовжки crypto_len.
1. Виділення відкритого ключа та *seed* з секретного

$$pk:=\text{subst}(sk, \text{sk\_len\_internal}, \text{pk\_len}),$$
$$seed:= \text{subst}(sk, \text{sk\_len\_internal}–\text{seed\_len}, \text{seed\_len}).$$

2. Розшифрування повідомлення (функція indcpa_dec)

$$\text{m\_calc:=indcpa\_dec}(ct, sk).$$

3. *buf*:=m_calc||*pk*.
4. Обчислення гешу для *buf*

$$kr:=G(buf).$$

5. *coins*:=*subst*(*kr*, seed_len, seed_len)
6. Зашифрування отриманої послідовності

$$\text{ct\_cmp:=indcpa\_enc}(buf, pk, coins).$$

7. Перевірка коректності.

success:=OK;

if *ct*≠ct_cmp then

success:=ERROR;

end if

8. Обчислення гешу зашифрованого повідомлення (функція *H*) та запис його в другу половину *kr*

kr:=subst(*kr*, 0, seed_len)||*H*(*ct*, crypto_len).

9. Коригування *kr* в залежності від успішності операції розшифрування

if success=ERROR then

kr:=*seed*;

end if

10. Формування загального секрету (функція KDF)

ss:= KDF (*kr*, 2*seed_len).

1.7.1. Розшифрування. Функція indcpa_dec

В х і д .

C – зашифроване повідомлення;

sk – секретний ключ.

В и х і д .

m – повідомлення.

1. Розпаковка *C*. Виділення вектора vec_u та полінома pol_v (функції vec_unpack^{du}, pol_unpack^{dv}).

2. Розпаковка секретного ключа. Обчислення вектора поліномів vec_s (функція vec_unpack^{q}).

3. Обчислення *mp*=pol_v–vec_s*vec_u.

4. Обчислення повідомлення *m*=poly_to_msg (mp).

2. Алгоритм Скеля

Алгоритм є NTRU подібним алгоритмом [4] з полем, визначеним в [5].

2.1. Параметри

Загальні параметри алгоритму наведено в табл. 3.

Таблиця 3

Загальні параметри алгоритму Скеля

Позначення	Призначення	Формула або значення
λ	Рівень криптостійкості	3, 5, 7
n	Степінь полінома. Визначає кількість його коефіцієнтів. Просте число, для якого поліном $x^n–x–1$ є незвідним	$\begin{cases} 881 & \lambda = 3 \\ 1201 & \lambda = 5 \\ 1471 & \lambda = 7 \end{cases}$
p	Менший модуль	*p*=3
q	Більший модуль, просте число, за яким зводять усі коефіцієнти полінома *R*/*q*	$\begin{cases} 7673 & \lambda = 3 \\ 9221 & \lambda = 5 \\ 12251 & \lambda = 7 \end{cases}$
t	Натуральне число, кількість ненульових елементів поліномадорівнює 2*t*	$\begin{cases} 159 & \lambda = 3 \\ 192 & \lambda = 5 \\ 255 & \lambda = 7 \end{cases}$
seed_len	Довжина *seed* (байтів)	$\begin{cases} 32 & \lambda = 3 \\ 48 & \lambda = 5 \\ 64 & \lambda = 7 \end{cases}$

У стандарті застосовують поліноми, позначені $R/3$, R/q. Поліном $R/3$ в якості коефіцієнтів застосовує значення {0, 1, –1}. Поліном R/q в якості коефіцієнтів застосовує цілі значення в інтервалі [0, q–1].

В якості алгоритмів гешування та потокового шифрування можна застосовувати довільні дозволені функції. Для обчислення тестових векторів в якості алгоритму гешування застосовується алгоритм Kupyna (ДСТУ 7564:2014), алгоритму потокового шифрування – Струмок (ДСТУ 8845:2019). У подальшому функції гешування позначені: H256, H512, H. H256 – результат гешування 256 біт, H512 – результат гешування 512 біт, H – результат гешування 256 (для λ=3) та 512 біт (для λ=5, 7).

2.2. Ідентифікатор алгоритму

Алгоритм застосовує ідентифікатор – рядок байтів завдовжки 3 байти. Позначимо цей ідентифікатор OID. Усі функції алгоритму застосовують один ідентифікатор.

2.3. Допоміжні функці

2.3.1. Генерація випадкових та псевдовипадкових послідовностей

Для генерації послідовностей застосовують довільні дозволені алгоритми. Для обчислення тестових векторів застосовують алгоритм ДСТУ 8845:2019 [3]. Перед застосуванням алгоритму треба ініціалізувати генератор випадковими ключем та вектором ініціалізації (функція RandomInit). Довжина ключа визначається рівнем криптостійкості і дорівнює 256 бітів для λ=3 та 512 – для λ=5, 7. Вектор ініціалізації завжди має довжину 256 бітів. Для обчислення ключа (*key*) та вектора ініціалізації (*iv*) для тестових векторів застосовуються два 64-бітних числа *val*1, *val*2. Стан генератора визначається параметром *ctx*, який змінюється після кожного застосування генератора:

$$key := H(val1, 8); \qquad iv := H256(val2, 8).$$

Замість значень ключа та вектора ініціалізації для ініціалізації генератора можна застосовувати рядок байтів (*b*) зазначеної довжини (b_len) (функція PsevdoRandomInit). У цьому випадку

$$\text{Ключ} = H(b, \text{b\_len}).$$

Для генерації вектора ініціалізації застосовують: *OID* (3 байти), останній байт в *b* (*blast*, самий правий байт), рядок з 28 байтів – нулів (zero28)

$$\text{Вектор ініціалізації} = H256(OID||blast||\text{zero28}).$$

В подальшому функція для генерації наступної послідовності позначена gen_posl. Функція gen_posl в якості параметрів приймає *ctx* та довжину послідовності в байтах та повертає в якості результату *ctx* та сформовану послідовність.

2.3.2. Алгоритми пакування

Алгоритми пакування застосовують для перетворення поліномів різних типів в рядок октетів, зворотніх перетворень та для перетворення рядка октетів в цілі невід'ємні числа при заданій кількості бітів для одного числа.

При формуванні бітового рядка передбачається, що біт 0 – самий лівий біт байту (в алгоритмі Kyber біт 0 – самий правий біт байту).

2.3.2.1. Пакування R/3 поліномів з заданою кількістю ненульових елементів (pack^3, unpack^3)

1. Виділяється бітовий рядок *bs* завдовжки $\left\lceil \frac{2t}{8} \right\rceil * 8 + \left\lceil \frac{n}{16} \right\rceil * 16$.

2. В перші $\left\lceil \frac{2t}{8} \right\rceil * 8$ бітів рядка *bs* записується 0, якщо наступний ненульовий коефіцієнт дорівнює –1, та 1, якщо 1.

3. В бітовий рядок *bs*1 записується 0, якщо відповідний коефіцієнт дорівнює 0, та 1 інакше.

Довжина рядка байтів для *R*/3 полінома дорівнює $\left\lceil \frac{2t}{bits\_in\_byte} \right\rceil + 2 * \left\lceil \frac{n}{2*bits\_in\_byte} \right\rceil$.

Для параметрів алгоритму $pol3\_len = \begin{cases} 152 & \lambda = 3 \\ 200 & \lambda = 5. \\ 248 & \lambda = 7 \end{cases}$

2.3.2.2. Пакування R/3 поліномів (pack^{2_3}, unpack^{2_3}) в рядок байтів для довільної кількості ненульових елементів

Цей алгоритм перетворює два сусідніх коефіцієнти полінома в три біта і навпаки відповідно до табл. 4.

Таблиця 4

Кодування коефіцієнтів R/3 полінома

Бітовий рядок	Коефіцієнти полінома	Бітовий рядок	Коефіцієнти полінома
000	0, 0	100	1, 1
001	0, 1	101	1, -1
010	0, -1	110	-1, 0
011	1, 0	111	-1, 1

Довжина рядка байтів для *R*/3 полінома дорівнює $\left\lceil \frac{3(n+1)}{2* bits\_in\_byte} \right\rceil$.

Для параметрів алгоритму $pol3\_len = \begin{cases} 166 & \lambda = 3 \\ 226 & \lambda = 5\cdot \\ 276 & \lambda = 7 \end{cases}$

При перетворенні полінома в рядок байтів (функція pack^{2_3}) можлива помилка, якщо два сусідніх елементи полінома, починаючи з елементу з парним номером, дорівнюють -1, тобто, знайдеться i, для якого $c[2i]$=-1, $c[2i+1]$=-1.

2.3.2.3. Пакування R/q поліномів (pack^{q}, unpack^{q})

Коефіцієнти полінома записуються в бітовий рядок один за одним. Під кожний коефіцієнт відводиться $\log_2 q$ біт.

Довжина рядка байтів для *R*/q полінома дорівнює $\left\lceil \frac{n*\log_2 q}{bits\_in\_byte} \right\rceil$.

Для параметрів алгоритму $polq\_len = \begin{cases} 1432 & \lambda = 3 \\ 2102 & \lambda = 5. \\ 2575 & \lambda = 7 \end{cases}$

2.3.2.4. Пакування R/q поліномів по модулю 4 (pack^{4}, unpack^{4})

Коефіцієнти полінома беруться по модулю 4 та записуються в бітовий рядок один за одним. Під кожний коефіцієнт відводиться 2 біт.

Довжина рядка байтів для *R*/q полінома дорівнює $\left\lceil \frac{2n}{bits\_in\_byte} \right\rceil$.

Для параметрів алгоритму $pol4\_len = \begin{cases} 221 & \lambda = 3 \\ 301 & \lambda = 5\cdot \\ 368 & \lambda = 7 \end{cases}$

2.3.2.5. Розпакування рядка байтів (unpack^{1}, unpack^{w})

Кількість бітів для кожного числа дорівнює 1 або визначається значенням параметра n і приймає значення

$$w = \begin{cases} 11 & \lambda = 3\ (n = 881) \\ 12 & \lambda = 5\ (n = 1201). \\ 12 & \lambda = 7\ (n = 1471) \end{cases}$$

Із заданого рядка байтів обираються наступні 1 або w бітів, починаючи з самого лівого (біту 0). Ця бітова послідовність розглядається як ціле невід'ємне число, яке і є наступним елементом масиву.

2.3.3. Генерація поліномів

2.3.3.1. Генерація R/3 полінома із заданою кількістю ненульових елементів. Функція gen_r3t_pol

Нехай кількість ненульових елементів дорівнює *T*.

Функція генерує псевдовипадковий рядок, в якому спочатку розташовані біти для визначення знаків ненульових елементів (всього $\left\lceil \frac{T}{bits\_in\_byte} \right\rceil$ байтів), а потім – для індексів ненульових елементів, всього потрібно *T* індексів. Для кожного індексу відводиться $w=\lceil \log_2 n \rceil+1$ біт. Для забезпечення однакової імовірності усіх індексів значення поточного числа для *i*-го індексу, яке перевищує $2^w-(2^w \bmod i)$, ігнорується. Ось чому довжина послідовності для генерації індексів перевищує необхідну. Для генерації випадкової перестановки застосовується алгоритм Рональда Фішера (Ronald Fisher) і Франка Йетса (Frank Yates) [6].

В х і д .

ctx – контекст генератора;

T – визначає кількість ненульових елементів.

В и х і д .

ctx – контекст генератора;

r3_pol – *R*/3 поліном.

1. Визначення довжини псевдовипадкового рядка для визначення індексів ненульових елементів та їх знаків. Для індексів виділяється подвійна пам'ять

$$\text{len\_sign}:=\left\lfloor \frac{T}{bits\_in\_byte} \right\rfloor + 1;\ \text{len\_ind}:=2 * \left\lceil \frac{\log_2 n*T}{bits\_in\_byte} \right\rceil.$$

2. Генерація псевдовипадкового рядка для знаків та індексів

buf:=gen_posl (len_sign+len_ind)

sign_buf:=subst (Buf, 0, len_sign);

ind_buf:=subst (Buf, len_sign, len_ind);

3. Перетворення sign_buf в масив чисел {0 або 1}

sign, count_sign:= unpack[1] (sign_buf, len_sign);

4. Створення випадкової послідовності

r3_pol=0 {*i* = 0..*n*–1} Початкова ініціалізація

k=0, *i*=*n*–*T*, *j*=0, *w*=$\log_2 n$+1, max_ind=2^w

while *k*<*T* do

ind, count_ind:=unpack^w (ind_buf, len_ind);

while *i*<*n* та *j*<count_ind та *k*<*T* do

ind=ind[*j*];

j=*j*+1;

if ind<max_ind–max_ind mod *i* then

ind = ind mod *i*;

r3_pol [*i*] = r3_pol [ind]; r3_pol [ind] = 1

if sign [*k*] == 0 then

r3_pol [ind] = -1

end if

k=*k*+1

i=*i*+1

end if

end while

if *k*!= T then Обробка закінчення послідовності ind

len_ind := $\left\lceil \frac{2 * (T - k) * w}{bits\_in\_byte} \right\rceil$

ind_buf, *ctx*:=gen_posl (*ctx*, len_ind)

end if

end while

2.3.3.2. Генерація R/3 полінома з довільною кількістю ненульових елементів. Функція gen_r3_pol

Спочатку генерують псевдовипадковий рядок байтів для обчислення коефіцієнтів полінома. Якщо значення наступного байту ≥243 (3^5), цей октет ігнорують, в іншому разі його застосовують для формування значень п'яти коефіцієнтів, а саме – значення октету переводять в трирічну систему числення, тобто формують відповідні значення $b_4b_3b_2b_1b_0$, які записують як відповідні коефіцієнти полінома в форматі {-1, 0, 1}.

П р и к л а д .

Нехай потрібно визначити коефіцієнти полінома, які відповідають октету 0x56. Шістнадцятковому значенню 56 відповідає десяткове значення 86, що дорівнює 10012 в трирічній системі. Отримуємо відповідні значення коефіцієнтів полінома a_4=1, a_3=0, a_2=0, a_1=1, a_0=-1.

Імовірність появи значень ≥243 за рівноімовірного генератора становить (256–243)/256, тобто менше 6 %. Для забезпечення достатньої кількості сформованих значень рекомендують формувати кількість октетів в два рази більше, ніж потрібно.

2.3.4. Обчислення загального секрету (ss) та частини інкапсульованого ключа (C)

В х і д .
r – *R*/3 поліном;
ss_len – довжина *ss*.
В и х і д .
C – частина інкапсульованого ключа, рядок байтів;
C_len – довжина *C*.
ss – загальний секрет (рядок байтів).
1. Пакування полінома *r*
temp, temp_len := pack^3(*r*)
2. Обчислення *C*, *ss*
if λ=3 then
temp:=H512 (temp, temp_len)
C_len:=32;
C:=subst (temp, 0, seed_len);
ss:= subst (temp, seed_len, ss_len)
else
*temp*1:=*temp*||1; *temp*2:=*temp*||2
C:= H512 (*temp*1, temp_len+1)
C_len := 64;
*temp*2 := H512 (*temp*2, temp_len+1)
ss:= subst (*temp*2, 0, ss_len)
end if

2.4. Генерація ключів (gen_keys)

В х і д .
Немає.
В и х і д .
sk – секретний ключ завдовжки $\left\lceil \frac{2t+n}{bit\_byte} \right\rceil$,
pk – відкритий ключ завдовжки $\left\lceil \frac{n*\log_2 q}{bit\_byte} \right\rceil$.
1. Ініціалізація генератора випадкових чисел
ctx:=RandomInit()
2. Генерація полінома *G*
G, *ctx*:= gen_r3t_pol (*ctx*, 2**n*/3+1)

3. success:=ERROR
4. while success=ERROR do Цикл генерації полінома *f*
Генерація полінома *F*
F, ctx:= gen_r3t_pol (*ctx*, 2**t*)
Обчислення полінома *f*
f:=(1+*pF*) mod *q*
Обчислення інверсії для *f* в полі $x^n–x–1$
*f*_1, success := inverse (*f*)
end while
5. Генерація відкритого ключа
h:=*g***f*_1 в полі $x^n–x–1$
6. Формування *pk*, *sk*
pk:=pack^q (*h*); *sk* := pack^3 (*F*)

$$\text{sk\_len} = \begin{cases} 152 & \lambda = 3 \\ 200 & \lambda = 5 \\ 248 & \lambda = 7 \end{cases} \qquad \text{pk\_len} = \begin{cases} 1432 & \lambda = 3 \\ 2102 & \lambda = 5 \\ 2575 & \lambda = 7 \end{cases}.$$

2.5. Алгоритм інкапсуляції

У процесі виконання алгоритму формується випадкове повідомлення завдовжки не менше 32 байтів і не більше, ніж pol3_len–seed_len–1 та дорівнює $\max\_msg\_len = \begin{cases} 133 & \lambda = 3 \\ 177 & \lambda = 5 \\ 211 & \lambda = 7 \end{cases}$. Для усіх параметрів max_msg_len не перевищує 255, тому для запису довжини повідомлення відводиться один байт.

В х і д .
pk – відкритий ключ одержувача;
ss_len – загальний секрет, довжина, байтів.
В и х і д .
success – успішність операції, OK – для успішної операції, ERROR – навпаки;
ss – загальний секрет – рядок байтів, довжина якого не перевищує seed_len;
cc – інкапсульований ключ (рядок байтів).
1. Перевірка параметра ss_len
if ss_len <= seed_len
success := OK
else
success := ERROR
end if
2. Продовження, якщо *success*=OK
if *success* = OK then
3. Розпакування відкритого ключа та генерація випадкового повідомлення
h:=unpack^q (*pk*); msg, msg_len := gen_msg ();
success:=ERROR
4. while *success*=ERROR do Цикл шифрування
5. Формування рядка *M*
M:=*seed*||msg_len||*msg*||*zero*
де:
seed – псевдовипадковий рядок завдовжки seed_len;
msg, msg_len – повідомлення для шифрування (*msg*) та його довжина (seed_len≤msg_len≤max_msg_len);
zero – рядок байтів с кодом 0 завдовжки max_msg_len–msg_len.
6. Перетворення рядка *M* в *R*/3 поліном

MTrin:=unpack$^{2-3}$(*M*)

7. Формування рядка *s* та його довжини

s_len:=3+m_len+2*seed_len

s:=*OID*||*m*||*seed*||*subst*(*pk*, 0, seed_len)

8. Формування *R*/3 полінома з 2*t* ненульовими елементами згідно з рядком *s*

ctx:=PsevdoInitRandom (*ctx*, *s*, s_len)

r:=gen_r3t_pol (*ctx*, 2*t*)

9. Обчислення

R:=*r***h* в полі *R*/*q*; *R*4:= pack^{4} (*R*);

mask:= gen_r3_pol (*R*4);

*pol*3:=(*MTrin*+*mask*) mod *p*

10. Перевірка умови за кількістю (ненульових (*k*1), нульових (*n*–*k*1)) елементів полінома *pol*3

if *k*1>=2**t* та *n*–*k*1>=*t* then

success=OK

end if

11. end while

12. Результат шифрування

c:=(*R*+*pol*3) mod *q*; *cc*:=pack^{q} (*c*)

13. Обчислення загального секрету та частини інкапсульованого ключа

ss, *C*, C_len:=gen_C_ss (*r*)

14. Обчислення інкапсульованого ключа

cc:=*C*||*cc*; cc_len:=C_len+polq_len

end if

2.6. Алгоритм декапсуляції

В х і д .

cc – інкапсульований ключ;

sk – секретний ключ отримувача;

pk – відкритий ключ отримувача.

ss_len – довжина загального секрету (байтів).

В и х і д .

success – ознака успішності;

ss – загальний секрет

1. Перевірка коректності довжини загального секрету ss_len.

success:=ERROR

if success ≤seed_len then

2. Розпакування секретного ключа, відкритого ключа та окремих компонентів *cc*.

F:=unpack^{3}(*sk*); *h*:=unpack^{q}(*pk*); *f*=*p***F*+1

3. Розпакування окремих компонентів *cc*.

if λ=3 then C_len:=32; else C_len:=64 end if

C'=subst (*cc*, 0, C_len); *c*'=subst (*cc*, C_len, polq_len);

e:=unpack^{q}(*c*')

4. Розшифрування

a'=*e***f* в полі *R*/*q* *pol*3:=*a*'mod *p*

5. Перевірка умови за кількістю (ненульових (*k*1), нульових (*n*–*k*1)) елементів полінома *pol*3.

success:=ERROR;

if *k*1>=2**t* та *n*–*k*1>=*t* then

success=OK

end if

6. Відновлення рядка *M*.
if *success*=OK then
R:=(*e*–*pol*3) mod *q*; *R*4:= pack^{4} (*R*);
mask:=gen_r3_pol (*R*4);
MTrin:=*pol*3–*mask*; *M*, *success*:=pack$^{2-3}$(*MTrin*)
end if
7. В разі успішного завершення відновлення рядка байтів *M* та перевірка коректності усіх полів.
if *success*=OK then
success=ERROR;
seed:=subst (*M*, 0, seed_len);
msg_len:=subst (*M*, seed_len, 1);
if msg_len≥*seed* та msg_len≤max_msg_len then
msg:= subst (*M*, seed_len+1, msg_len);
zero_count := max_msg_len–msg_len;
zero':=subst (*M*, seed_len+1+msg_len, zero_count);
if zero_count байтів *zero*' дорівнюють 0 then
success=OK;
end if
endif
8. У разі успішного завершення відновлення рядка байтів *s* та полінома *r*
if *success*=OK then
s:=*OID*||*msg*||*seed*||subst (*pk*, 0, seed_len)
s_len:=3+msg_len+seed_len+seed_len
ctx:=PsevdoInitRandom (*ctx*, *s*, s_len)
r:=gen_r3t_pol (*ctx*, 2*t*)
9. Обчислення полінома *R*' та порівняння його з *R*.
R'=*r***h*
if *R*'≠*R* then
success:= ERROR;
end if
end if
10. У разі успішного завершення обчислення *ss*
if *success* = OK then
ss, *C*, C_len:=gen_C_ss (*r*)
if *C*'≠*C* then
success:=ERROR;
end if
end if

3. Порівняння алгоритмів

3.1. Порівняння розмірів ключів та інкапсульованих даних

Для порівняння застосовується загальний режим при λ=3, 5.

Таблиця 5

Порівняння розмірів ключів та інкапсульованих даних

Алгоритм	λ=3			λ=5		
	pk_len	sk_len	Cc_len	pk_len	sk_len	Cc_len
Kyber	1184	2400	1088	1568	3168	1568
Скеля	1432	152	1464	2102	200	2166

Порівняння розмірів показує, що розмір секретного ключа для алгоритму Скеля суттєво менший за розмір секретного ключа для алгоритму Kyber, ніж у випадку, якщо разом

з секретним ключом зберігати відкритий, як це робиться в алгоритмі Kyber. Розміри відкритого та інкапсульованого ключа для алгоритму Kyber менші.

3.2. Порівняння швидкодії

Порівняння швидкодії виконується для криптостійкостей, які підтримуються для обох алгоритмів, тобто λ=3, 5.

Для визначення швидкодії алгоритму Kyber застосовується реалізація авторів алгоритму Kyber (round 3, Optimized_Implementation).

Для визначення швидкодії алгоритму Скеля застосовується реалізація авторів статті.

Характеристика обладнання для експерименту:

11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz.

Microsoft Visual C++ 2019 00435-00000-00000 AA349.

Mode=64 bit.

Таблиця 6

Порівняння швидкодії

Алгоритм	λ=3			λ=5		
	GenKeys	Incaps	Decaps	GenKeys	Incaps	Decaps
Kyber	127018	139615	160693	216923	232294	234565
Скеля	605320	59532	75849	1048405	82969	105700

Час генерації ключів для Скелі перевищує час генерації ключів для Kyber, але у той же час у алгоритмі Kyber відсутня операція інверсії, а у Скелі вона є, і вона є досить ресурсоємною.

Висновок

При збільшенні довжини відкритого ключа та результату інкапсуляції алгоритм Скеля більш ефективний, ніж алгоритм Kyber для інкапсуляції та декапсуляції.

Список літератури:

1. Post-Quantum Cryptography. [Електронний ресурс]. Режим доступу: https://csrc.nist.gov/.

2. ДСТУ 8961:2019 Інформаційні технології. Криптографічний захист інформації. Алгоритми асиметричного шифрування та інкапсуляції ключів. Режим доступу: http://online.budstandart.com/ua/catalog/doc-page.html?id_doc=88056.

3. DSTU8845. [Електронний ресурс]. Режим доступу: https://github.com/outspace/dstu8845.

4. NTRU. A submission to the NIST post-quantum standardization effort. [Електронний ресурс]. Режим доступу: https://ntru.org/.

5. NTRU Prime. [Електронний ресурс]. Режим доступу: https://ntruprime.cr.yp.to/.

6. D. Knuth The Art of Computer Programming vol. 2. 3rd. Boston : Addison–Wesley, 1998. P. 145–146.

Надійшла до редколегії 05.08.2022

Відомості про авторів:

Горбенко Іван Дмитрович – д-р техн. наук, професор, Харківський національний університет імені В. Н. Каразіна, професор кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук, АТ "Інститут Інформаційних Технологій", головний конструктор, Україна; e-mail: gorbenkoi@iit.kharkov.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», начальник відділу програмування; Україна; e-mail: iit@iit.kharkov.ua; ORCID: https://orcid.org/0000-0001-9249-0497

Єсіна Марина Віталіївна – канд. техн. наук, Харківський національний університет імені В.Н. Каразіна, доцент кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; науковий співробітник-консультант АТ «Інститут Інформаційних технологій»; Україна; e-mail: m.v.yesina@karazin.ua; ORCID: https://orcid.org/0000-0002-1252-7606

Пономар Володимир Андрійович – канд. техн. наук, Харківський національний університет імені В.Н. Каразіна, науковий співробітник кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; Україна; e-mail: Laedaa@gmail.com; ORCID: https://orcid.org/0000-0001-5271-2251