УДК 004.056.5 **DOI:10.30837/rt.2021.3.206.04**

Ю.І. ГОРБЕНКО, д-р техн. наук, О.Г. КАЧКО, канд. техн. наук, С.О. КАНДІЙ

ДОСЛІДЖЕННЯ ДОЦІЛЬНОСТІ ЗАСТОСУВАННЯ AVX512 ДЛЯ РЕАЛІЗАЦІЇ СУЧАСНИХ АЛГОРИТМІВ ЕЛЕКТРОННИХ ПІДПИСІВ

Вступ

Постквантова криптографія є напрямом досліджень, що вивчає криптографічні перетворення, які захищені від атак з використанням квантових комп'ютерів. У 2016 р. NIST США оголосили про початок конкурсу NIST PQC, метою якого є створення нових постквантових криптографічних стандартів. Наразі триває третій фінальний етап цього конкурсу. Згідно з аналізом спеціалістів NIST [8], одним з перспективних напрямів у постквантовій криптографії є криптографія на алгебраїчних решітках. У звіті [7] зазначається , що NIST планує стандартизувати хоча б один електронний підпис (ЕП) на решітках. Серед електронних підписів фіналістами , які є представниками криптографії на решітках, є CRYSTALS-Dilithium [2] та Falcon[3].

Сучасні ЕП на решітках потребують ефективних обчислень у поліноміальних кільцях вигляду $R_q = \mathbf{Z}_q[X]/(f(X))$, де $f(X)$ – деякий незвідний поліном. Під час генерації ключової пари, вироблення та верифікації підпису основними операціями є складання та множення поліномів. Складання поліномів потребує лінійну кількість обчислень і працює достатньо швидко. Множення поліномів «шкільним» методом потребує квадратичну кількість операцій, що робить реалізації неефективними. Тому для ефективної реалізації операції множення часто використовується теоретико-числове перетворення (NTT) [4]. Для підвищення швидкодії на сучасних процесорах можливо використовувати векторизовані (SIMD) набори інструкцій. Серед існуючих реалізацій найчастіше використовуються AVX2 інструкції [2 – 4]. У той же час можливість використання AVX512 інструкцій залишається малодослідженою.

Мета роботи – дослідження доцільності використання AVX512 інструкцій для оптимізації FFT і NTT, що використовуються у сучасних ЕП на алгебраїчних решітках. Зокрема, в роботі наведений метод реалізації теоретико-числового перетворення з використанням AVX512 для ЕП CRYSTALS-Dilithium та Falcon. Показано збільшення швидкодії порівняно з еталонними оптимізованими авторськими реалізаціями.

Особливості реалізації теоретико-числового перетворення

У переважної більшості ЕП на решітках [1 – 3] використовується циклотомічне кільце $R_q = \mathbf{Z}_q[X]/(X^n+1)$. Якщо $q \equiv 1 (\mathrm{mod}\, 2n)$, то $\mathbf{Z}_q$ містить n примітивних $2n$-х коренів з одиниці. У цьому випадку R_q є полем розкладу полінома X^n+1 і, відповідно до китайської теореми про залишки, має місце гомоморфізм:

$$f \mapsto (f(\xi), f(\xi^3), ..., f(\xi^{2n-1})) : \mathbf{Z}_q[X]/(X^n+1) \to \prod_i \mathbf{Z}_q[X]/(X-\xi^i), \quad (1)$$

Для описаного випадку цей гомоморфізм є ізоморфізмом [4]. Теоретико-числове перетворення полягає у обчисленні цього ізоморфізму. Операція множення поліномів замінюється на покомпонентне множення векторів над полем $\mathbf{Z}_q$. Відповідно повна операція множення поліномів матиме наступний вигляд:

$$f * g = NTT^{-1}(NTT(f) * NTT(g)), \quad (2)$$

NTT можливо обчислити за $O(n \log n)$ операцій за допомогою алгоритму швидкого теоретико-числового перетворення [4] (рис. 1).

Вхідні данні:

Поліном $f = c_0 + c_1 X \dots c_{n-1} X^{n-1} \in \mathbb{Z}_q[X]/(X^n + 1)$

Примітивні корені з одиниці: $\zeta_k \quad 0 \le k < n.$

Алгоритм 1. Швидке теоретико-числове перетворення

$k \leftarrow 1$
for $l \leftarrow n/2; l > 0; l \leftarrow l/2$ **do**
  for $s \leftarrow 0; s < n; s \leftarrow j + l$ **do**
    for $j \leftarrow s; j < s + l; j \leftarrow j + 1$ **do**
      $t \leftarrow \zeta_k \cdot c_{j+l}$
      $c_{j+l} \leftarrow c_j - t$
      $c_j \leftarrow c_j + t$
    end for
    $k \leftarrow k + 1$
  end for
end for

Алгоритм 2. Зворотне швидке теоретико-числове перетворення

$k \leftarrow 0$
for $l \leftarrow 1; l < n; l \leftarrow 2l$ **do**
  for $s \leftarrow 0; s < n; s \leftarrow j + l$ **do**
    for $j \leftarrow s; j < s + l; j \leftarrow j + 1$ **do**
      $t \leftarrow c_j$
      $c_j \leftarrow t + c_{j+l}$
      $c_{j+l} \leftarrow t - c_{j+l}$
      $c_{j+l} \leftarrow \zeta_k \cdot c_{j+l}$
    end for
    $k \leftarrow k + 1$
  end for
end for
for $j \leftarrow 0; j < n; j \leftarrow j + 1$ **do**
  $c_j \leftarrow c_j / n$
end for

Рис. 1. Алгоритм швидкого теоретико-числового перетворення

Алгоритми на кожній ітерації обчислюють відображення

$$\begin{aligned} & f \mapsto (f \bmod X^{n/2} - \xi^{n/2},\ f \bmod X^{n/2} + \xi^{n/2}) \\ & \mathbf{Z}_q[X]/(X^n - \xi^n) \to \mathbf{Z}_q[X]/(X^{n/2} - \xi^{n/2}) \times \mathbf{Z}_q[X]/(X^{n/2} + \xi^{n/2}) \end{aligned}, \tag{3}$$

При цьому коефіцієнти відповідних поліномів на кожній ітерації обчислюються як

$$\begin{aligned} c'_i &= c_i + \xi^{n/2} c_{n/2+i} \\ c''_i &= c_i - \xi^{n/2} c_{n/2+i} \end{aligned}, \tag{4}$$

Через N ітерацій відображення (3) буде тотожне (1).

Фіналісти третього етапу NIST PQC використовують модифіковані алгоритми обчислення NTT. Від обраних розробниками модифікацій залежать тестові вектори, тож при реалізації схеми необхідно враховувати ці зміни. Розглянемо детальніше особливості реалізації NTT для ЕП CRYSTALS-Dilithium [2] та Falcon [3].

Особливості NTT для ЕП Dilithium

Через те, що коефіцієнти поліномів лежать у кільці $\mathbf{Z}_q$, під час обчислень в стандартній реалізації відбувається багато приведень за модулем q. Ця операція є дорогою. Щоб зменшити кількість її використань при множенні коефіцієнтів автори використовують редукцію

Монтгомері, яка залежить від фактора r і дозволяє швидко обчислювати вирази виду $a * b * r^{-1} \bmod q$. Автори використовують $r = 2^{32}$. На фазі передобчислень всі примітивні корені з одиниці заносяться до масиву. Кожен корень помножується на фактор $2^{-32} \bmod q$. Відповідно після NTT перетворення коефіцієнти поліному будуть помножені на цей фактор. Під час покомпонентного множення цей фактор зникає.

Іншою оптимізацією є використання лінивої редукції. В стандартній реалізації після кожного складання коефіцієнтів під час перетворення NTT також необхідно приводити за модулем q, як і у випадку множення коефіцієнтів. Редукцію Монтгомері тут застосувати неможливо. Проте лінива редукція дозволяє уникнути важких операцій приведення за модулем. Оскільки під час складань значення зростає доволі повільно, то коефіцієнти можна не приводити за модулем у кожній ітерації, а приводити лише в кінці.

На кожній ітерації фактично масив можливо розглядати як сукупність блоків. Ілюстрація такого представлення наведена на рис. 2.

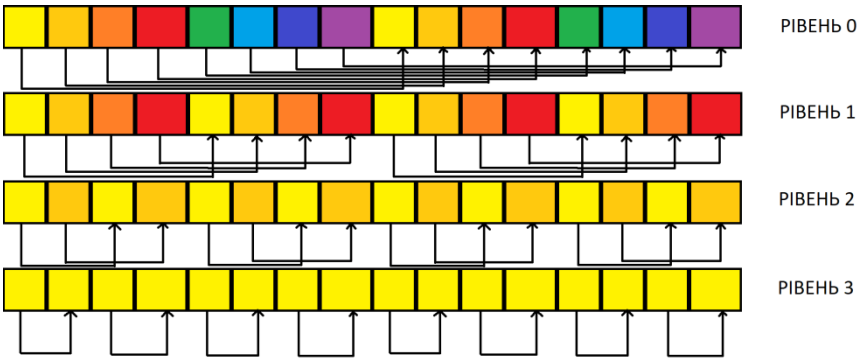

Рис. 2. Представлення взаємодії коефіцієнтів поліномів як взаємодії блоків. Кожен рівень є ітерацією алгоритму швидкого NTT

При обробці кожен блок взаємодіє лише з одним іншим блоком, що знаходиться справа. Відповідно буде 2,4,8,16,32,64,128,256 блоків розміру 128,64,32,16,8,4,2,1 елементів на відповідних ітераціях. З врахуванням всіх покращень перетворення (4) матиме вигляд:

```
t = montgomery_reduce((uint64_t)zeta * c[j + len]);
c[j + len] =( c[j] + 2*Q – t);
c[j] = (c[j] + t);
```

де len – розмір поточного блоку, zeta – примітивний корінь, c – масив з коефіцієнтами поліному.

При використанні AVX512 на перших ітераціях розмір блоків занадто великий, щоб їх повністю розмістити в регістрах zmm. Відповідно, стратегія обчислень буде відрізнятися. Кожен коефіцієнт зберігається в 32-бітній змінній. Проте через використання лінивої редукції з кожною ітерацією значення буде швидко зростати і для зберігання проміжкових даних необхідно 64 біти. Тобто, один регістр zmm може зберігати 8 коефіцієнтів. Припустимо, що у регістрах zmm0=left_block0, zmm1= left_block1, zmm2= left_block2, zmm3=left_block3 зберігаються відповідні значення c[j], а в zmm4=right_block0,zmm5= right_block1,zmm6= right_block2,zmm7= right_block3 відповідні значення c[j + len]. Значення відповідних примітивних коренів відповідно у регістрах zmm8=zeta0,zmm9=zeta1,zmm10=zeta2,zmm11=zeta3 і константа 2Q у регістрі zmm9. Тоді операцію (4) з врахуванням лінивої редукції та редукції Монтгомері можливо реалізувати наступним чином:

- Обчислюємо zeta * c[j + len]);
  right_block0 = _mm512_mul_epu32(right_block0, zeta0);
  right_block1 = _mm512_mul_epu32(right_block1, zeta1);

right_block2 = _mm512_mul_epu32(right_block2, zeta2);
right_block3 = _mm512_mul_epu32(right_block3, zeta3);

- Обчислюємо t = montgomery_reduce((uint64_t)zeta * c[j + len]);

zmm12 = _mm512_mul_epu32(right_block0, q-1);
zmm13 = _mm512_mul_epu32(right_block1, q-1);
zmm14 = _mm512_mul_epu32(right_block2, q-1);
zmm15 = _mm512_mul_epu32(right_block3, q-1);
zmm12 = _mm512_mul_epu32(zmm12, q);
zmm13 = _mm512_mul_epu32(zmm13, q);
zmm14 = _mm512_mul_epu32(zmm14, q);
zmm15 = _mm512_mul_epu32(zmm15, q);
zmm12 = _mm512_add_epi64(zmm12, right_block0);
zmm13 = _mm512_add_epi64(zmm13, right_block1);
zmm14 = _mm512_add_epi64(zmm14, right_block2);
zmm15 = _mm512_add_epi64(zmm15, right_block3);
zmm12 = _mm512_srli_epi64(zmm12, 32);
zmm13 = _mm512_srli_epi64(zmm13, 32);
zmm14 = _mm512_srli_epi64(zmm14, 32);
zmm15 = _mm512_srli_epi64(zmm15, 32);

- Обчислюємо c[j + len] =(c[j] + 2*Q – t);

right_block0= _mm512_add_epi32(left_block0, 2q);
right_block1= _mm512_add_epi32(left_block1, 2q);
right_block2= _mm512_add_epi32(left_block2, 2q);
right_block3= _mm512_add_epi32(left_block3, 2q);
right_block0= _mm512_sub_epi32(right_block0, zmm12);
right_block1= _mm512_sub_epi32(right_block1, zmm13);
right_block2= _mm512_sub_epi32(right_block2, zmm14);
right_block3= _mm512_sub_epi32(right_block3, zmm15);

- Обчислюємо c[j] = (c[j] + t);

left_block0 = _mm512_add_epi32(left_block0, zmm12);
left_block1 = _mm512_add_epi32(left_block1, zmm13);
left_block2 = _mm512_add_epi32(left_block2, zmm14);
left_block3 = _mm512_add_epi32(left_block3, zmm15);

На ітераціях, де розмір блоку є занадто великим для зберігання в регістрах можливо цю операцію викликати декілька разів для покриття повного розміру блоку.

Особливості NTT для ЕП Falcon

У ЕП Falcon реалізація перетворення NTT значно відрізняється від CRYSTALS-Dilithium. Значні зміни відбуваються через новаторський підхід при вирішенні NTRU рівняння. При пошуку рішення на кожному кроці обчислюється проекція поліному за допомогою норми на полі у деяке менше підполе. При цьому коефіцієнти поліному значно зростають, що призводить до необхідності використання довгої арифметики. Автори використовують систему залишкових класів для вирішення цієї проблеми [3]. Реалізація перетворення (4) приймає вигляд

```
uint32_t x, y;

x = *r1;
y = *r2;
*r1 = modp_add(x, y, p);
*r2 = modp_montymul(
        modp_sub(x, y, p), s, p, p0i);
```

де r1,r2 є вказівниками на відповідні елементи в блоках; p – одне з простих чисел, що використовується у системі залишкових класів у якості модуля; p0i – зворотнє до числа p.

При цьому вказівники r1, r1 змінюються з деяким кроком в залежності від підполя, в якому відбуваються перетворення. Це ускладнює використання векторизованих інструкцій так як данні неможливо зчитати одним блоком. Тож, для використання векторизованих інструкцій необхідно спочатку перегрупувати данні в залежності від підполя. Після перегрупування можливо застосувати такий же підхід, як і у випадку CRYSTALS-Dilithium. При цьому операцію складання за модулем можливо реалізувати як

```
static inline __m512i modp_add_avx512(__m512i a,__m512i b,__m512i p){
  __m512i tmp1,tmp2;

  tmp1 = _mm512_add_epi32(a,b);
  tmp1 = _mm512_sub_epi32(tmp1,p);
  tmp2 = _mm512_srli_epi32(tmp1,31);
  tmp2 = _mm512_sub_epi32(_mm256_set1_epi32(0),tmp2);
  tmp2 = _mm512_and_si256(tmp2,p);
  tmp1 = _mm512_add_epi64(tmp1,tmp2);
  return tmp1;
}
```

Операцію віднімання відповідно

```
static inline __m512i modp_sub_avx512(__m512i a,__m512i b,__m512i p){
  __m512i tmp1,tmp2;

  tmp1 = _mm512_sub_epi32(a,b);
  tmp2 = _mm512_srli_epi32(tmp1,31);
  tmp2 = _mm512_sub_epi32(_mm256_set1_epi32(0),tmp2);
   tmp2 = _mm512_and_si256(tmp2,p);
  tmp1 = _mm512_add_epi64(tmp1,tmp2);
  return tmp1;
}
```

І редукцію Монтгомері як

```
static inline __m512i modp_montymul_avx512(__m512i a,__m512i b,
                  __m512i p,__m512i p0i){
  __m512i tmp1,tmp2,tmp3;
  __m512i mask = _mm512_set1_epi64x(0x7FFFFFFF);

  tmp1 =_mm512_mul_epu32(a,b);
  tmp2 =_mm512_mul_epu32(tmp1,p0i);
  tmp2 =_mm512_and_si256(tmp2,mask);
  tmp2 =_mm512_mul_epu32(tmp2,p);

  tmp1 = _mm512_add_epi64(tmp1,tmp2);
  tmp1 = _mm512_srli_epi64(tmp1,31);
  tmp1 = _mm512_sub_epi32(tmp1,p);
  tmp2 = _mm512_srli_epi64(tmp1,31);
  tmp2 = _mm512_sub_epi32(_mm256_set1_epi32(0),tmp2);
  tmp2 = _mm512_and_si512(tmp2,p);
  tmp1 = _mm512_add_epi64(tmp1,tmp2);
  return tmp1;
}
```

В порівнянні з CRYSTLAS-Dilithium, реалізація NTT є складнішою і потребує більшої кількості ресурсів.

Множення в спектральній області з використанням AVX512

Множення в спектральній області, відповідно до формул (2) та (3), відбувається покомпонентно. Розглянемо загальну ситуацію, де потрібно знайти значення виразу

$$a_1 * b_1 + ... + a_l * b_l, \tag{5}$$

де $a_1, ..., a_l, b_1, ..., b_l$ є поліноми у NTT, що представлені (в спектральній області).

Щоб не використовувати дорогу операцію приведення за модулем, скористаємося редукцією Монтгомері: кожен елемент помножимо на фактор 2^{32} і виконаємо операцію редукції у кінці. Тож, для обчислення виразу (5) виконаємо наступні кроки:

- Оскільки вираз (5) є сумою, то необхідні регістри, що слугуватимуть у ролі акумулятора. Відведемо під цю роль регістри zmm2-zmm9.
- Нехай a та b є вказівниками типу _m512i на відповідні представлення поліномів в пам'яті. Зчитаємо з пам'яті поточний блок

```
zmm10 = _mm512_load_si512(a);
    zmm12 = _mm512_load_si512(a + 1);
    zmm14 = _mm512_load_si512(a + 2);
    zmm16 = _mm512_load_si512(a + 3);
    zmm18 = _mm512_load_si512(b);
    zmm20 = _mm512_load_si512(b + 1);
    zmm22 = _mm512_load_si512(b + 2);
zmm24 = _mm512_load_si512(b + 3);
```

- Помножимо на фактор 2^{32} :

```
    zmm11 = _mm512_srli_epi64(zmm10, 32);
    zmm13 = _mm512_srli_epi64(zmm12, 32);
    zmm15 = _mm512_srli_epi64(zmm14, 32);
    zmm17 = _mm512_srli_epi64(zmm16, 32);
    zmm19 = _mm512_srli_epi64(zmm18, 32);
    zmm21 = _mm512_srli_epi64(zmm20, 32);
    zmm23 = _mm512_srli_epi64(zmm22, 32);
zmm25 = _mm512_srli_epi64(zmm24, 32);
```

- Виконаємо по компонентне множення

```
zmm10 = _mm512_mul_epu32(zmm18, zmm10);
    zmm11 = _mm512_mul_epu32(zmm19, zmm11);
    zmm12= _mm512_mul_epu32(zmm20, zmm12);
    zmm13= _mm512_mul_epu32(zmm21, zmm13);
    zmm14 = _mm512_mul_epu32(zmm22, zmm14);
    zmm15 = _mm512_mul_epu32(zmm23, zmm15);
    zmm16 = _mm512_mul_epu32(zmm24, zmm16);
    zmm17 = _mm512_mul_epu32(zmm25, zmm17);
```

- Запишемо результати до обраних акумуляторів

```
    zmm2 = _mm512_add_epi64(zmm2, zmm10);
zmm3 = _mm512_add_epi64(zmm3, zmm11);
zmm4 = _mm512_add_epi64(zmm4, zmm12);
zmm5 = _mm512_add_epi64(zmm5, zmm13);
zmm6 = _mm512_add_epi64(zmm6, zmm14);
zmm7 = _mm512_add_epi64(zmm7, zmm15);
zmm8 = _mm512_add_epi64(zmm8, zmm16);
zmm9 = _mm512_add_epi64(zmm9, zmm17);
```

- На останньому кроці виконаємо редукцію Монтгомері над акумуляторами (в регістрах zmm0,zmm1 містяться відповідні константи для редукції.

```
zmm10 = _mm512_mul_epu32(zmm2, zmm0);
zmm11 = _mm512_mul_epu32(zmm3, zmm0);
zmm12 = _mm512_mul_epu32(zmm4, zmm0);
zmm13 = _mm512_mul_epu32(zmm5, zmm0);
zmm14 = _mm512_mul_epu32(zmm6, zmm0);
zmm15 = _mm512_mul_epu32(zmm7, zmm0);
zmm16 = _mm512_mul_epu32(zmm8, zmm0);
zmm17 = _mm512_mul_epu32(zmm9, zmm0);
zmm10 = _mm512_mul_epu32(zmm10, zmm1);
zmm11 = _mm512_mul_epu32(zmm11, zmm1);
zmm12 = _mm512_mul_epu32(zmm12, zmm1);
zmm13 = _mm512_mul_epu32(zmm13, zmm1);
zmm14 = _mm512_mul_epu32(zmm14, zmm1);
zmm15 = _mm512_mul_epu32(zmm15, zmm1);
zmm16 = _mm512_mul_epu32(zmm16, zmm1);
zmm17 = _mm512_mul_epu32(zmm17, zmm1);
zmm2 = _mm512_add_epi64(zmm10, zmm2);
zmm3 = _mm512_add_epi64(zmm11, zmm3);
zmm4 = _mm512_add_epi64(zmm12, zmm4);
zmm5 = _mm512_add_epi64(zmm13, zmm5);
zmm6 = _mm512_add_epi64(zmm14, zmm6);
zmm7 = _mm512_add_epi64(zmm15, zmm7);
zmm8 = _mm512_add_epi64(zmm16, zmm8);
zmm9 = _mm512_add_epi64(zmm17, zmm9);
zmm2 = _mm512_srli_epi64(zmm2, 32);
zmm4 = _mm512_srli_epi64(zmm4, 32);
zmm6 = _mm512_srli_epi64(zmm6, 32);
zmm8 = _mm512_srli_epi64(zmm8, 32);
```

Швидкодія реалізації NTT з використанням AVX512

Результати реалізації [5] наведено в таблиці для процесору Intel (R) Core (TM) i 9-7960X CPU @ 2.80GHz, 2808 MHz cores, 16 logical processors: 32. Компиляторы: Microsoft Visual Studio Community 2019, Version 16.6.2, VisualStudio.16.Release / 16.6.2 + 30204.135, / o2 optimization flags GCC 9.3, флаги оптимізації: -O3 –march=native,-mtune=native

Таблиця 1

Швидкодія теоретико-числового перетворення

	Linux, Dilithium (lang asm, GCC)	Linux(lang C, GCC)	Windows 10, (lang C, msvc)	Speedup Linux	Speedup Windows
ntt	987	663	727	1.48	1.35
Покомпонентне множення	135	81	87	1.6	1.51
Зворотнє ntt	972	677	741	1.43	1.31

В таблиці наведено результати виміру кількості тактів для кожної функції. Для виміру кількості тактів застосовувалась команда процесора rdtsc або відповідна функція. В другій колонці таблиці наведені результати для авторської реалізації (розробники Dilithium), які отримані з застосуванням ключів компілятора GCC, які застосовували автори (Linux).

В третій колонці наведені результати нашої реалізації для тих же ключів компілятору GCC, які застосовувалися для отримання попередніх результатів (Linux). В четвертій колонці наведено результати нашої реалізації з застосуванням компілятору msvc В наступних колонках наведено прискорення для нашої реалізації по відношенню до авторської реалізації.

Висновки

Розроблено реалізацію теоретико-числового перетворення з використанням AVX512 для постквантових електронних підписів, що є фіналістами конкурсу NIST PQC. Показано, що використання AVX512 дозволяє отримати прискорення у 1,5 рази, порівняно з існуючими реалізаціями. Для схеми Falcon реалізація NTT має складнішу реалізацію, ніж для CRYSTALS-Dilithium.

Список літератури:

1. Gorhan Alagic Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process. NISTIR 8309 / Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner

2. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler and Damien Stehlé CRYSTALS-Dilithium: Algorithm Specifications and Supporting Documentation. – Access mode: https://pq-crystals.org/dilithium/data/dilithium-specification.pdf

3. Thomas Prest et Al. aFlcon: Fast-Fourier Lattice-basedCompact Signatures over NTRU – Access mode: https://falcon-sign.info/falcon.pdf

4. Gregor Seiler Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography – Access mode: https://crypto.ethz.ch/publications/files/Seiler18.pdf

5. AVX512 NTT implementation for Dilithium. Access mode: https://github.com/KandiyIIT/dilithium_ntt_avx512

6. Качко О.Г. Осика О.Ф. Використання SIMD команд для паралельних обчислень. Навчальний посібник з дисципліни Паралельне програмування. Харків : ХНУРЕ, 2020. 274 с.

7. NISTR 8309. Status Report on the Second Round of the NIST Post-Quantum Cryptography Standartization Process. NIST, 2020. 39 p.

8. NIST Post-Quantum Cryptography Standartization Project : веб сайт. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization (дата звернення: 27.11.2020)

Надійшла до редколегії 13.09.2021

Відомості про авторів:

Горбенко Юрій Іванович – канд. техн. наук, АТ «інститут інформаційних технологій», перший заступник головного конструктора; Україна; e-mail: gorbenkou@iit.kharkov.ua; ORCID: https://orcid.org/0000-0003-0073-9107

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

Кандій Сергій Олегович – Харківський національний університет імені В. Н. Каразіна, аспірант кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; АТ «Інститут Інформаційних технологій», технік-конструктор; Україна; e-mail: sergeykandy@gmail.com; ORCID: https://orcid.org/0000-0003-0552-8341