$$\begin{tikzcd}
z \in \mathbb{C}^{\frac{N}{2}} \arrow[r, dash, "\Delta"] \arrow[d, dash, "\times A"'] & pt \in R_Q \arrow[r, dash] \arrow[d, dash] & ct \in R_Q^2 \arrow[d, "\text{lineartransform}"] \\
Az \in \mathbb{C}^{\frac{N}{2}} \arrow[r, dash, "\Delta^2"] & pt_1 \in R_Q \arrow[r, dash] & ct_1 \in R_Q^2
\end{tikzcd}$$

shift

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix},$$

$$Az = \underbrace{\begin{bmatrix} 1 \\ 6 \\ 11 \\ 16 \end{bmatrix}}_{\substack{V_0 \\ [shift=0]}} \odot \underbrace{\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix}}_{z} + \underbrace{\begin{bmatrix} 2 \\ 7 \\ 12 \\ 13 \end{bmatrix}}_{\substack{V_1 \\ [shift=1]}} \odot \underbrace{\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_0 \end{bmatrix}}_{rot(z)} + \underbrace{\begin{bmatrix} 3 \\ 8 \\ 9 \\ 14 \end{bmatrix}}_{\substack{V_2 \\ [shift=2]}} \odot \underbrace{\begin{bmatrix} z_2 \\ z_3 \\ z_0 \\ z_1 \end{bmatrix}}_{rot^2(z)} + \underbrace{\begin{bmatrix} 4 \\ 5 \\ 10 \\ 15 \end{bmatrix}}_{\substack{V_3 \\ [shift=3]}} \odot \underbrace{\begin{bmatrix} z_3 \\ z_0 \\ z_1 \\ z_2 \end{bmatrix}}_{rot^3(z)}$$

rotation of polynomial.

$$m(x) \longrightarrow z = \begin{bmatrix} m(\zeta) \\ m(\zeta^5) \\ m(\zeta^{5^2}) \\ m(\zeta^{5^3}) \end{bmatrix}$$

ζ: primitive-root with $x^N + 1 = 0$.

$$m(x^5) = rot(m)(x) \longrightarrow z^{rot} = \begin{bmatrix} m(\zeta^5) \\ m(\zeta^{5^2}) \\ m(\zeta^{5^3}) \\ m(\zeta^{5^4}) = m(\zeta) \end{bmatrix}$$

$$m(x) = \sum_{j=0}^{N-1} m[j]\, x^j, \quad m^{rot}(x) = \sum_{j=0}^{N-1} m[j]\, x^{5j} = \sum_{j=0}^{N-1} \underbrace{\boxed{m[j](-1)^{A_j}}}_{m^{rot}[r_j]} x^{r_j}$$

$$\boxed{5j = N \cdot A_j + r_j, \quad r_j \in [0, N-1]}$$

```
template<int N, int L>
void rot( const uint64_t q[L], const uint64_t m[L][N], int r,
                                         uint64_t mrot[L][N] ){
    if(r==0){ for(int i=0; i<L; i++)
              for(int j=0; j<N; j++)  mrot[i][j] = m[i][j]; }
    else{
        uint64_t temp[L][N];
        for( int i=0; i<L; i++)
        for ( int j=0; j<N; j++){  uint64_t Aj = (5*j) / N;
                                   uint64_t rj = (5*j) % N;
              temp[i][rj] = (Aj%2==0)? m[i][j] : ((q[i]-m[i][j])%q[i]);
        }
        rot<N,L>(q, temp, r-1, mrot);
    }
}
```

$m^{rot} = rot^r(m)$

temp = rot(m)

recursive call

rotation of ciphertext:

$$ct = [ct_0(x), ct_1(x)] \quad \text{with} \quad pt(x) = ct_0(x) + ct_1(x) * s(x)$$

Then

$$pt^{rot}(x) = pt(x^5) = ct_0^{rot}(x) + ct_1^{rot}(x) * s^{rot}(x).$$

$$ct^{rot} = [ct_0^{rot}, ct_1^{rot}] \xrightarrow[\text{decoding}]{s^{rot}} pt^{rot}$$

$\Downarrow$ key switching from s^{rot} to s

$$ks_{s^{rot} \to s}\left([ct_0^{rot}, ct_1^{rot}]\right) \to ct^{rot}$$

$$\vec{v}_i \odot rot(z) \xrightarrow{\Delta^2} \text{Encoding}_\Delta(\vec{v}_i) * pt \in R_Q \;—\; \text{Encoding}_\Delta(\vec{v}_i) * ct^{rot} \in R_Q^2$$

(Hadamard product) (convolution on pt) (convolution on ct)

```
template <int N, int L, int DNUM, int K, int NumDigs>
void rkey_gen( const SparseComplexMatrix<N/2, NumDigs>& A,
               const uint64_t q[L],
               const uint64_t p[K],
               const int      s[N],
               uint64_t rkey[NumDigs][DNUM][2][DNUM*K+K][N]){

    for(int i=0; i< NumDigs; i++){
        int Srot[N];                                 Srot = rot^shift (s)
        rot<N>(s, A.shift[i], Srot);
        swkgen<N,L,DNUM>(Srot, s, q, p, rkey[i]);
    }
}
```

```
template <int N> void rot(const int s[N], int r, int s^rot[N]){
    if(r==0) memcpy(s^rot, s, sizeof(int)*N);
    else{ int temp[N];
          for(int j=0; j<N; j++){ int Aj=(5*j)/N, bj=(5*j)%N;      temp=rot(s)
               temp[bj]= (Aj%2==0)? s[j] : -s[j];
          rot<N>(temp, r-1, s^rot);                                recursive call
    }
}
```

```
template<int N, int logN, int L, int DNUM, int K, int NumDigs>
```

level L ≤ DNUM*K
(full level

```
void lineartransform( const SparseComplexMatrix<N/2, DumDigs>& A,
                      const uint64_t    Delta,
                      const uint64_t    q[L],
                      const uint64_t    p[K],
                      const uint64_t    r̂Key[NumDigs][DNUM][2][DNUM*K+K][N],
                      const uint64_t    ĉt[2][L][N],
                            uint64_t    rês[2][L][N] ){
    uint64_t ct[2][L][N];
    for(int i=0; i<2; i++)
    for(int j=0; j<L; j++)
    for(int k=0; k<N; k++){
        ct[i][j][k] = ĉt[i][j][k];
        rês[i][j][k] = 0;
    }
    intt<N,L>(g, ct[0]);
    intt<N,L>(g, ct[1]);
```

res = 0

ct = intt(ĉt)

```
    for(int d=0; d<NumDigs; d++){
        uint64_t pt[L][N];
        encode<N, logN, L>(A.diagr[d], A.diagi[d], Delta, q, pt);
```

A.diag[d] $\xrightarrow{\Delta}$ pt

```
        uint64_t temp[2][L][N];
        rot<N,L>(q, ct[0], A.shift[d], temp[0]); ntt<N,L>(g, temp[0]);
        rot<N,L>(q, ct[1], A.shift[d], temp[1]); ntt<N,L>(g, temp[1]);
```

rotation by
A.shift[d]
times

```
        uint64_t ĉt^rot[2][L][N];
        ks<N, L, DNUM, K>(q, p, r̂Key[d], temp, ĉt^rot);
```

key-switching

```
        ntt<N,L>(g, pt)
        for(int i=0; i<2; i++)
        for(int j=0; j<L; j++)
        for(int k=0; k<N; k++)
            rês[i][j][k] = ( rês[i][j][k] + mul_mod(pt[j][k], ĉt^rot[i][j][k],
                                                     q[j])) % q[j];
    }
}
```

$\hat{pt} \odot \hat{ct}^{rot}$
$\downarrow \oplus$
$\hat{res}$

// test_lineartransform

$$A=\begin{array}{c} \begin{array}{ccc} \text{shift}=0 & \text{shift}=1 & \text{shift}=\frac{N}{2}-1 \end{array} \\ \begin{bmatrix} 2 & -1 & \cdots & -1 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ -1 & & \ddots & \end{bmatrix} \end{array}$$

tri-diagonal matrix

```
ooo

void main(){

    Z ooo
    pt ooo
    ct ooo
    ------------------------------------------
    SparseComplexMatrix<N/2,3> A;
    A.shift[0]=0;
      //  [1]=1;
      //  [2]= N/2-1;
    for(int i=0; i<3; i++)
    for(int j=0; j<N/2; j++){
        A.diagr[i][j]= ((double)rand())/RAND_MAX;
        A.diagi[i][j]= 0;
    }
    ------------------------------------------
    uint64_t r̂key[3][DNUM][2][DNUM*K+K][N];
    rkey_gen<N,L,DNUM,K,NumDigs>(A,q,p,s,r̂key);
    uint64_t r̂es[2][L][N];
    lineartransform<N,logN,L,DNUM,K,3>(A,Delta,q,p,r̂key,ĉt,r̂es);
    ------------------------------------------
    uint64_t r̂es_rs[2][L-1][N];
    RS_hat<N,L>(q, r̂es[0], r̂es_rs[0]);
      //        (q, r̂es[1], r̂es_rs[1]);
uint64_t pti[L-1][N];
    ------------------------------------------
    dec<N,L-1>(r̂es_rs, q, s, pti);
    double er[N/2], ei[N/2];
    decode<N,logN,L-1>(pti, Delta, q, er, ei);
    for(int i=0; i<N/2; i++){
        int im1 = (i==0)? (i-1+N/2): (i-1);
        int ip1 = (i==N/2-1)? (i+1-N/2): (i+1);
        er[i]-= A.diagr[0][i] * zr[i] + A.diagr[1][i]*zr[ip1] + A.diagr[2][i] *zr[im1];
        ei[i]-=       //      * zi[i]      //      zi[ip1] + //   zi[im1];
    }
    ooo
}
```