EvalMod Algorithm

Assume the case of $N=2^{10}$, MDNUM=3,

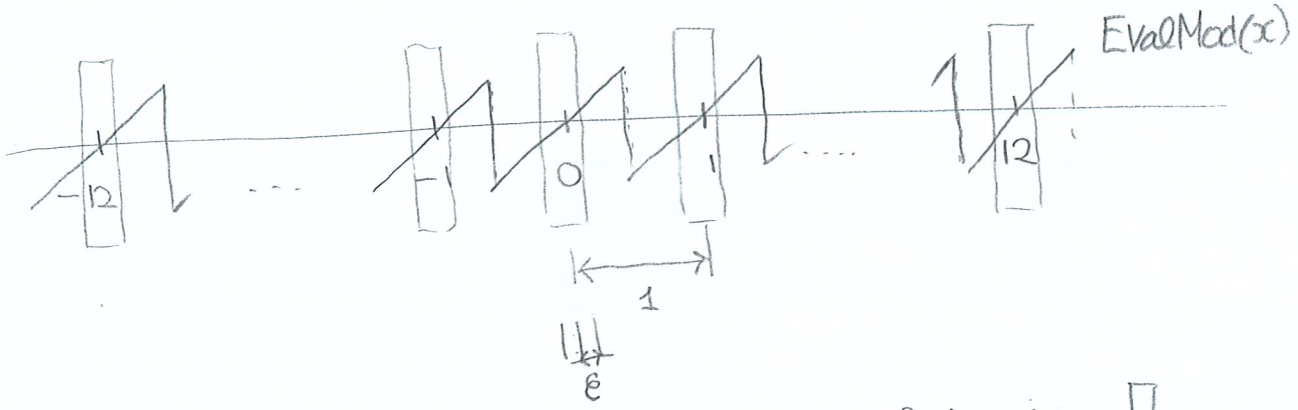
$L=24$, DNUM=3, $K=5$, $h=64$, $\Delta \cong 2^{50}$, $q \cong 2^{60}$.

$$z \in \mathbb{C}^{\frac{N}{2}} \xrightarrow{\Delta} pt \in R_q \longrightarrow ct \in R_q^2$$

$$\downarrow \qquad\qquad \downarrow \qquad\qquad \downarrow \text{modUp}$$

$$\xrightarrow{\Delta} pt+qI \in R_Q \longrightarrow ct \in R_Q^2$$

When $h=64$, the integer polynomial I is almost surely bounded by 12. When $\|z\|_\infty \le 1$, $\|pt\|_\infty \le \Delta$ and $\left\|\frac{pt}{q}\right\|_\infty \le \epsilon = \frac{1}{2^{10}} \cong 10^{-3}$.



Each $\left(\frac{pt}{q}+I\right)$ belongs to one of the strips $[]$, and the integer polynomial I can be removed by $\text{EvalMod}\left(\frac{pt}{q}+I\right) = \frac{pt}{q}$. Using $\text{EvalMod}(x) \cong \frac{\sin(2\pi x)}{2\pi}$,

$$\frac{\sin(2\pi x)}{2\pi} = \frac{\sin(\pi x)}{\pi} \cdot \cos(\pi x)$$

$$= p_1\left(\frac{x}{2}\right) \cdot q_1\left(\frac{x}{2}\right)$$

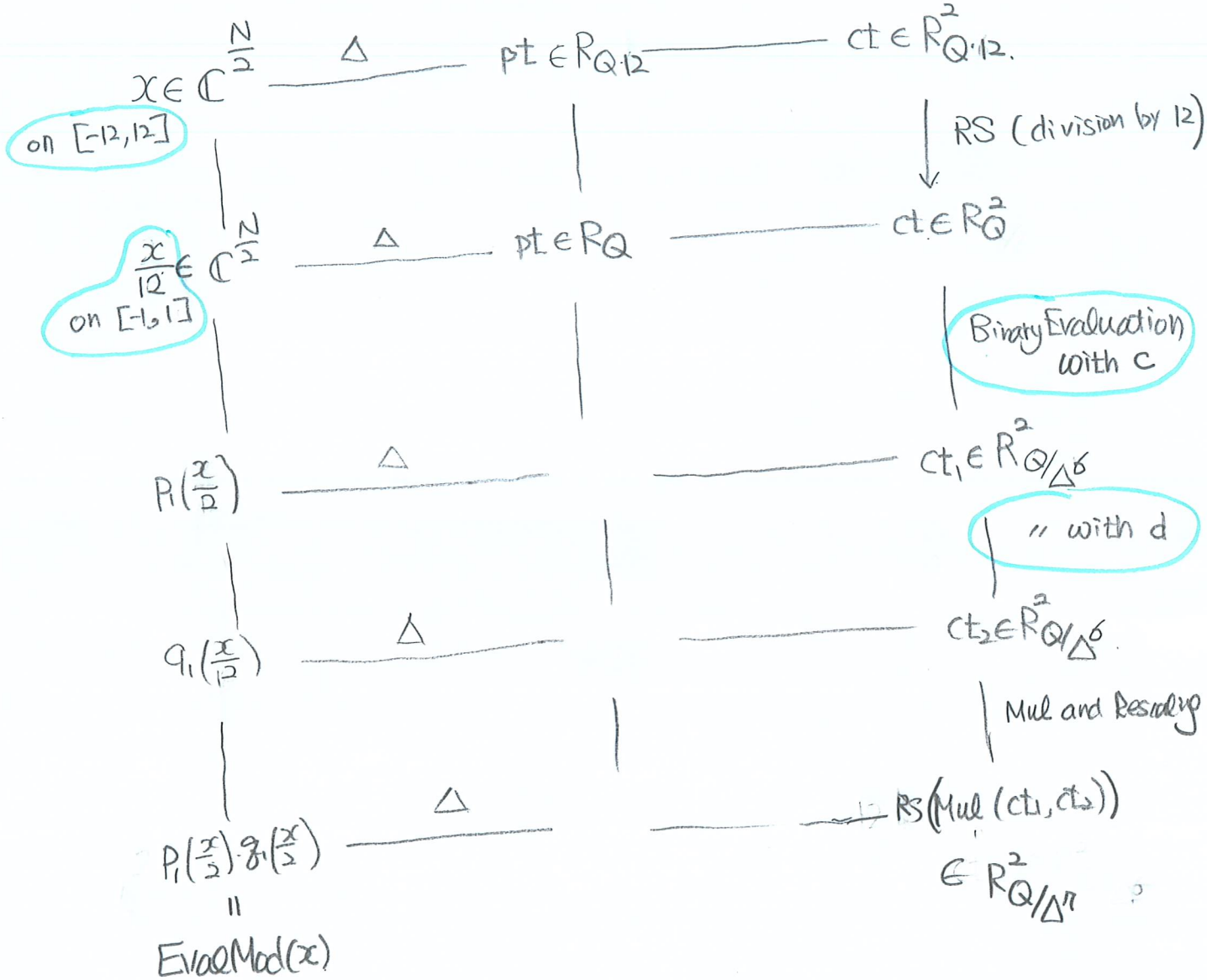
Let us define $p_1(x) \cong \frac{\sin(2\pi x)}{2\pi}$ on $[-6,6]$ with error 3.21×10^{-11} and,
$q_1(x) \cong \cos(2\pi x)$ on $[-6,6]$ with error 2.16×10^{-10} then,

$$\text{EvalMod}(x) \cong p_1\left(\frac{x}{2}\right) q_1\left(\frac{x}{2}\right)$$

$$P_1(x) = \sum_{i=0}^{63} c[i]\, T_i\left(\frac{x}{\delta}\right) \xrightarrow{\text{to binary evaluation}} c[0] + c[1]\,T_1\left(\frac{x}{\delta}\right) + T_2\left(\frac{x}{\delta}\right)\left[c[2] + \cdots\right]$$

Chebyshev polynomial defined on $[-1,1]$

$$q_1(x) = \sum_{i=0}^{63} d[i]\, T_i\left(\frac{x}{\delta}\right) \longrightarrow d[0] + d[1]\,T_1\left(\frac{x}{\delta}\right) + T_2\left(\frac{x}{\delta}\right)\left[d[2] + \cdots\right].$$



EvaoMod evaluation details

```
template<int N, int L, int DNUM, int K>
void EvalMod_h_64(const uint64_t p[L],
                  const uint64_t q[K], uint64_t Delta,
                  const uint64_t evk[DNUM][2][DNUM*K+K][N],
                  const uint64_t ct[2][L][N],
                  uint64_t ct_evalmod[2][L-7][N]) {
  double   c[64] = {...};
  double   d[64] = {...};
  uint64_t ct1[2][L-6][N];
  uint64_t ct2[2][L-6][N];

  eval_poly_deg64<N,L,DNUM,K>(q,p,c,Delta,ct,evk,ct1);

  eval_poly_deg64<N,L,DNUM,K>(q,p,d,Delta,ct,evk,ct2);

  mul_rs<N,L-6,DNUM,K>(q,p,evk,ct1,ct2,ct_evalmod);

}
```

test_evalmod.cpp

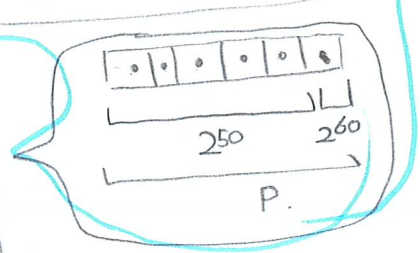
parameter choice set.

```
#define LOGN 10      #define DNUM 4      #define Delta (1ULL<<50)
#define L 24         #define K 6         #define N (1<<LOGN)
```

```
void main() {
```

q:

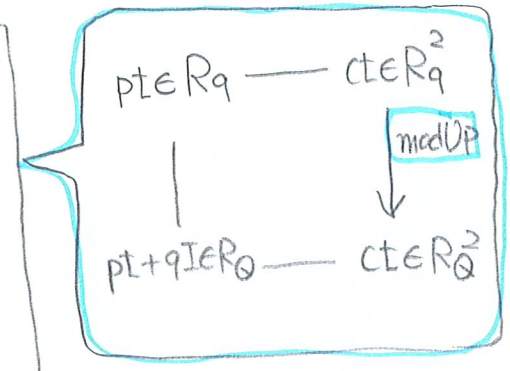

```
uint64_t q[L]  uint64_t p[K];
{
    uint64_t q50[20]; find_RNS_primes<20>(1ULL<<50, N, q50);
    uint64_t q60[9];          //    <9>(1ULL<<60, N, q60);
    q[0]=q60[0]; q[ ]=q60[ ], p[5]=q60[8];
    for(int i=1; i<=13; i++) q[i] = q50[i-1];
    for(int i=14; i<20; i++) q[i] = q60[ 1+i-14];
    q[21]= q50[13];
    for(int i=0; i<=4; i++) p[i]= q50[14+i];
    find_RNS_primes<1>((1ULL<<50)*12, N, &q[22]);
    q[23]= q50[19];
}
```




```
double zr[N/2], zi[N/2]; ooo
int h=64; ooo
uint64_t pt[1][N]; ooo
uint64_t ct_bottom[2][1][N]; ooo.
```

the default encoding & encryption

```
uint64t ct[2][L][N];
for(int i=0; i<2; i++)
for(int j=0; j<N; j++)
    ct[i][0][j] = ct_bottom[i][0][j];
intt<N,1>(g, ct[0]);   intt<N,1>(g, ct[1]);
modUp<N,L>(g,1, ct[0]); modUp<N,L>(g,1, ct[1]);
ntt<N,L>(g, ct[0]);    ntt<N,L>(g, ct[1]);
```



```
SparseComplexMatrix<N/2, 27> A[3];
splitUOR_logN_10(A);
uint64_t  evk^                  [DNUM][2][DNUM*K+K][N];
uint64_t  cKey^                 [  "              "  ];
uint64_t  rKey^  [3][27]        [  "              "  ];
swkgen_logN_10<L,DNUM,K>(q, p, s, A, evk^, cKey^, rKey^);
uint64_t  ct^_cts [2][2][L-3][N];
CoeffToSlot_logN_10<L,DNUM,K>(q, p, Delta, A, cKey^, rKey^, ct^_cts);
```

(annotation above the last argument: $\hat{ct}$,)

CoeffToSlot

$\frac{1}{12}\left(\frac{pt}{q}+I\right) \in \left(\mathbb{C}^{\frac{N}{2}}\right)^2 \xrightarrow{\;q\;} \longrightarrow \overrightarrow{ct_{cts}} \in \left(R^2_{Q_{L-3}}\right)^2$

```
uint64_t  ct^_evalmod [2][2][L-10][N];

EvalMod_h_64<N, L-3, DNUM, K>(q, p, q[0], evk^, ct^_cts[0], ct^_evalmod[0]);
    "       (       "      , ct^_cts[1], ct^_evalmod[1]);
```

$\vec{x} = \left(\frac{pt}{q}+I\right)^{bitReversed}$

$\left(\frac{pt}{q}\right)^{bitReversed} \Rightarrow$

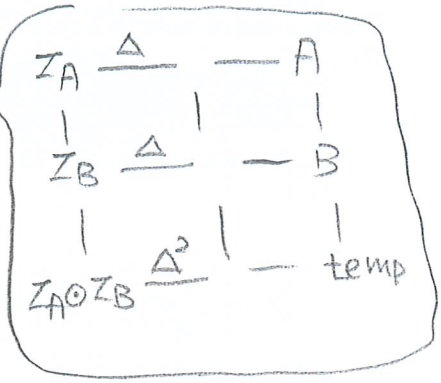
$\overrightarrow{\left(\frac{pt}{q}\right)}^{bitReversed} \xrightarrow{\;q\;} \longrightarrow \overrightarrow{ct_{evalmod}}$

included in CKKS_poly.h

multiplication and rescaling

```
template<int N, int L, int DNUM, int K>
void mul_rs(const uint64_t q[L],
            const uint64_t p[K],
            const uint64_t evk[DNUM][2][DNUM*K+K][N],
            const uint64_t Â[2][L][N],
            const uint64_t B̂[2][L][N],
                  uint64_t Ĉ[2][L-1][N]){
```

```
uint64_t temp[2][L][N];
mul<N,L,DNUM,K>(q,p,evk,Â,B̂,temp);
```




```
RS_hat<N,L>(q, temp[0], Ĉ[0]);
RS_hat<N,L>(q, temp[1], Ĉ[1]);
```

$Z_A \otimes Z_B$ —Δ— C, under the assumption $\Delta \simeq q[L-1]$

}