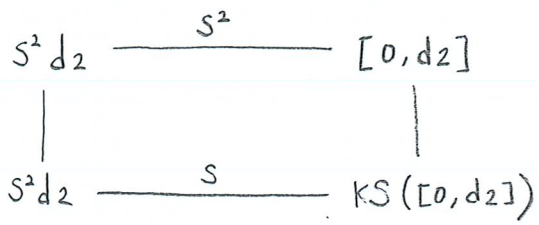
$Mul(A,B) = C$,

$A_0 + s \cdot A_1 = a$

$B_0 + s \cdot B_1 = b$

$\underbrace{A_0B_0}_{d_0} + s\underbrace{(A_0B_1 + A_1B_0)}_{d_1} + s^2\underbrace{A_1B_1}_{d_2} = ab$

$C = [d_0, d_1] + KS([0, d_2])$



```
template <int N, int L, int DNUM, int K>
void mul( 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][N]) {
  uint64_t d̂0[L][N];
  uint64_t d̂1[L][N];
  uint64_t d̂2[L][N];
  for (int i=0; i<L; i++)
  for (int j=0; j<N; j++) {
      d̂0[i][j] = mul_mod(Â[0][i][j], B̂[0][i][j], q[i]);
      d̂1[i][j] = (mul_mod(Â[0][i][j], B̂[1][i][j], q[i])
                 +mul_mod(Â[1][i][j], B̂[0][i][j], q[i])) % q[i];
      d̂2[i][j] = mul_mod(Â[1][i][j], B̂[1][i][j], q[i]);
  uint64_t temp[2][L][N];
  for(int i=0; i<L; i++)
  for(int j=0; j<N; j++) {
      temp[0][i][j] = 0;
      temp[1][i][j] = d̂2[i][j];
}
```

```
KS<N,L,DNUM,K>(q,P, evk, temp, ĉ);

for(int i=0; i<L; i++)
for(int j=0; j<N; j++){
    ĉ[0][i][j] = (ĉ[0][i][j] + d̂0[i][j]) %. q[i];
    ĉ[1][i][j] = (ĉ[1][i][j] + d̂1[i][j]) %. q[i];
}

}
```

Polynomial form: $p(x) = c[0] + c[1]x + \cdots + c[N-1]x^{N-1}$

This form is not used in CKKS, for x^{N-1} may be too large.

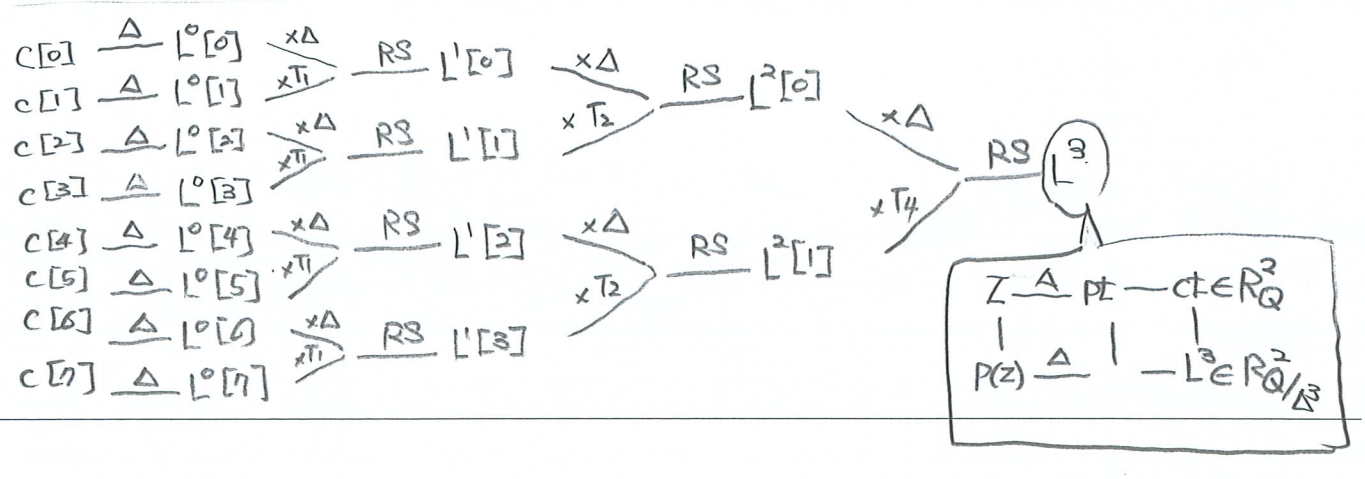
Instead, Chebyshev form is prefered.

$$p(x) = c[0] + c[1]T_1(x) + c[2]T_2(x) + \cdots,$$

Each $T_i(x)$ is bounded by 1, when $x \in [-1, 1]$.

To minimize multiplications, the Chebyshev form is reordered to form a binary tree.

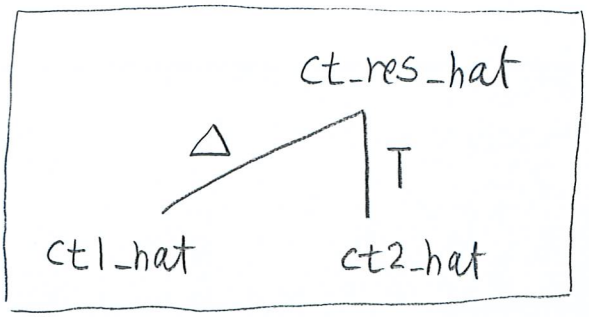
e.g. when $N=8$, $p(x) = (c[0] + c[1]T_1) + T_2(c[2] + c[3]T_1) + T_4[(c[4] + c[5]T_1) + T_2(c[6] + c[7]T_1)]$



```
template <int N, int L, int DNUM, int K>
void Chebyshev_doubling( const uint64_t q[L],
                         const uint64_t P[K], uint64_t Delta,
                         const uint64_t T[2][L][N],
                         const uint64_t evk_hat[DNUM][2][K*DNUM+K][N],
                               uint64_t Tnext[2][L-1][N] ) {

  uint64_t temp[2][L][N];

  mul<N,L,DNUM,K>(q, P, evk_hat, T, T, temp);

  for(int i=0; i<L; i++)
  for(int j=0; j<N; j++) {
    temp[0][i][j] = (2* temp[0][i][j]) %. q[i];
    temp[1][i][j] = (2* temp[1][i][j]) %. q[i];
  }
  RS_hat<N, L>(q, temp[0], Tnext[0]);
  RS_hat<N,L>(q, temp[1], Tnext[1]);

  uint64_t scale = Delta;
  for(int i=0; i<L-1; i++)
  for(int j=0; j<N; j++) {
    Tnext[0][i][j] = ( Tnext[0][i][j] + q[i] - (scale %. q[i]) ) %. q[i];
  }
}
```

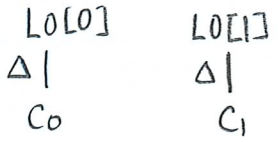
$$\text{Tnext} = RS(2\overbrace{T^2}^{\text{temp}}) - \Delta$$

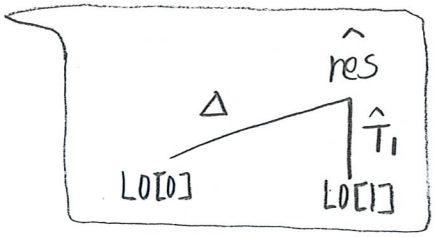
```
template <int N, int L, int DNUM, int K>
void eval-poly-merge( const uint64_t q[L],
                      const uint64_t P[K], uint64_t Delta,
                      const uint64_t T[2][L][N],
                      const uint64_t evk_hat[DNUM][2][DNUM*K+K][N],
                      const uint64_t ct1[2][L][N],
                      const uint64_t ct2[2][L][N],
                            uint64_t ct_res_hat[2][L-1][N]) {

  uint64_t temp[2][L][N];
  mul <N, L, DNUM, K> (q, P, evk_hat, T, ct2_hat, temp);
  for (int j=0; j<L; j++)
  for (int k=0; k<K; k++) {
    temp[0][j][k] = (temp[0][j][k] + mul_mod (ct1_hat[0][j][k], Delta % q[j], q[j]
                    % q[j];

    temp[1][j][k] = (temp[1][j][k] + mul_mod (ct1_hat[1][j][k], Delta % q[j], q[j])) % q[
  }
  RS_hat <N, L> (q, temp[0], ct_res_hat[0]);
  RS_hat <N,L> (q, temp[1], ct_res_hat[1]);
}
```



```
template <int N, int L>
void eval_poly_deg2 ( const uint64_t q[L],
                      const double   c[2], uint64_t Delta,
                      const uint64_t T̂1[2][L][N],
                      uint64_t res[2][L-1][N] ) {

  uint64_t LO[2][L][N];
  for(int i=0; i<2; i++)
  for(int j=0; j<L; j++) {
      for(int K=1; K<N; K++)
          LO[i][j][K] = 0;
      if (c[i] >= 0)   LO[i][j][K] = uint64_t (c[i] * Delta + .5) %. q[j];
      else             LO[i][j][K] = (q[j] - uint64_t ((-c[i]) * Delta + .5) %. q[j]);
  }
  ntt<N,L> (q, LO[0]);
  ntt<N,L> (q, LO[1]);

{
  uint64_t temp[2][L][N];
  for(int j=0; j<L; j++)
  for(int K=0; K<N; K++) {
      temp[0][j][K] = ( mul_mod ( LO[0][j][K], Delta %. q[j], q[j])
                      + mul_mod (LO[1][j][K], T̂1[0][j][K], q[j])) %. q[j];
      temp[1][j][K] = mul_mod (LO[1][j][K], T̂1[1][j][K], q[j]);
  }
  RS_hat <N,L> (q, temp[0], reŝ[0]);
  RS_hat <N,L> (q, temp[1], reŝ[1]);
}
;
```

LO[i] = encode(c[i], Δ);





```
template <int N, int L, int DNUM, int K>
void eval_poly_deg4(const uint64_t q[L],
                    const uint64_t p[K],
                    const double c[4], uint64_t Delta,
                    const uint64_t T1[2][L][N],
                    const uint64_t T2[2][L-1][N],
                    const uint64_t evk_hat[DNUM][2][K*DNUM+K][N],
                    uint64_t res_hat[2][L-2][N]) {

  uint64_t LI[2][2][L-1][N];
  eval_poly_deg2<N,L>(q, c, Delta, T1, LI[0]);
  eval_poly_deg2<N,L>(q, c+2, Delta, T1, LI[1]);

  eval_poly_merge<N, L-1, DNUM, K>(q, p, Delta, T2, evk_hat, LI[0], LI[1],
                                   res_hat);
}
```

```
template <int N, int L, int DNUM, int K>
void eval-poly-deg 8 ( const uint64_t q[L],
                       const uint64_t p[K],
                       const double   c[8], uint64_t Delta,
                       const uint64_t T1[2][L][N],
                       const uint64_t T2[2][L-1][N],
                       const uint64_t T4[2][L-2][N],
                       const uint64_t evk_hat[DNUM][2][K*DNUM+K][N],
                             uint64_t res_hat[2][L-3][N] ) {
  uint64_t L2[2][2][L-2][N];
  eval-poly-deg4 <N,L,DNUM,K> (q, p, c, Delta, T1, T2, evk_hat, L2[0]);
  eval-poly-deg4 <N,L,DNUM,K> (q,p, ct4, Delta, T1,T2, evk_hat, L2[1]);

  eval-poly-merge <N,L-2,DNUM,K> (q, p, Delta, T4, evk_hat, L2[0], L2[1], res_hat)
}
```

```
template <int N, int L, int DNUM, int K>
void eval-poly-deg16 (const uint64_t q[L],
                      const uint64_t P[K],
                      const double   C[16], uint64_t Delta,
                      const uint64_t T1 [2][L][N],
                      const uint64_t T2 [2][L-1][N],
                      const uint64_t T4 [2][L-2][N],
                      const uint64_t T8 [2][L-3][N],
                      const uint64_t evk_hat [DNUM][2][K*DNUM +K][N],
                            uint64_t res_hat [2][L-4][N] ) {

  uint64_t L3[2][2][L-3][N];
  eval-poly-deg8 <N, L, DNUM, K> (q, P, C, Delta, T1, T2, T4, evk_hat, L3[0]);
  eval-poly-deg8 <N, L, DNUM, K> (q, P, C+8, Delta, T1, T2, T4, evk_hat, L3[1]);

  eval-poly-merge <N, L-3, DNUM, K> (q, P, Delta, T8, evk_hat, L3[0], L3[1], res_hat);
}
```

```
template <int N, int L, int DNUM, int K>
void eval-poly-deg32 ( const uint64_t q[L],
                       const uint64_t p[K],
                       const double c[32], uint64_t Delta,
                       const uint64_t T1[2][L][N],
                       const uint64_t T2[2][L-1][N],
                       const uint64_t T8[2][L-2][N],
                       const uint64_t T16[2][L-3][N],
                       const uint64_t evk_hat[DNUM][2][K*DNUM+K][N],
                       uint64_t res_hat[2][L-5][N] ) {

  uint64_t L4[2][2][L-4][N];
  eval-poly-deg16<N,L,DNUM,K>(q, p, c, Delta, T1, T2, T8, evk_hat, L4[0]);
  eval-poly-deg16<N,L,DNUM,K>(q, p, c+16, Delta, T1, T2, T8, evk_hat, L4[1]);
  eval-poly-merge<N,L-4,DNUM,K>(q, p, Delta, T16, evk_hat, L4[0], L4[1], res_hat);
}
```

```
template <int N, int L, int DNUM, int K>
void eval_poly_deg64 (const uint64_t q[L],
                      const uint64_t p[K],
                      const double   c[64], uint64_t Delta,
                      const uint64_t T1 [2][L][N],
                      const uint64_t T2[2][L-1][N],
                      const uint64_t T4[2][L-2][N],
                      const uint64_t T8 [2][L-3][N],
                      const uint64_t T16 [2][L-4][N],
                      const uint64_t T32 [2][L-5][N],
                      const uint64_t evk_hat [DNUM][2][K*DNUM+K][N],
                      uint64_t res_hat [2][L-6][N]) {

  uint64_t L5 [2][2][L-5][N];
  eval_poly_deg32 <N,L,DNUM,K> (q, p, c, Delta, T1, T2, T4, T8, T16, evk_hat, L5[0]);
  eval_poly_deg32 <N,L,DNUM,K> (q, p, c+32, Delta, T1, T2, T4, T8, T16, evk_hat, L5[1]);

  eval_poly_merge <N, L-5, DNUM, K> (q, p, Delta, T32, evk_hat, L5[0], L5[1], res_hat);
}
```

```
template <int N, int L, int DNUM, int K>
void eval_poly_deg128( const uint64_t q[L],
                       const uint64_t P[K],
                       const double   c[128], uint64_t Delta,
                       const uint64_t T1[2][L][N],
                       const uint64_t T2[2][L-1][N],
                       const uint64_t T4[2][L-2][N],
                       const uint64_t T8[2][L-3][N],
                       const uint64_t T16[2][L-4][N],
                       const uint64_t T32[2][L-5][N],
                       const uint64_t T64[2][L-6][N],
                       const uint64_t evk_hat[DNUM][2][K*DNUM+K][N],
                             uint64_t res_hat[2][L-7][N] ) {

  uint64_t L6[2][2][L-6][N];

  eval_poly_deg64<N,L,DNUM,K>(q, P, c, Delta, T1, T2, T4, T8, T16, T32,
                              evk_hat, L6[0]);

  eval_poly_deg64<N,L,DNUM,K>(q, P, c+64, Delta, T1, T2, T4, T8, T16, T32,
                              evk_hat, L6[1]);

  eval_poly_merge<N,L-6,DNUM,K>(q, P, Delta, T64, evk_hat, L6[0], L6[1],
                                res_hat);
}
```

```
template <int N, int L, int DNUM, int K>
void eval-poly-deg256( const uint64_t q[L],
                       const uint64_t P[K],
                       const double   c[256], uint64_t Delta,
                       const uint64_t T1[2][L][N],
                       const uint64_t T2[2][L-1][N],
                       const uint64_t T4[2][L-2][N],
                       const uint64_t T8[2][L-3][N],
                       const uint64_t T16[2][L-4][N],
                       const uint64_t T32[2][L-5][N],
                       const uint64_t T64[2][L-6][N],
                       const uint64_t T128[2][L-7][N],
                       const uint64_t evk_hat[DNUM][2][K*DNUM+K][N],
                       uint64_t res_hat[2][L-8][N] ) {

uint64_t L7[2][2][L-7][N];

eval-poly_deg128 <N,L,DNUM,K> (q, P, c, Delta, T1,T2,T4,T8,T16,T32,T64,
                               evk_hat, L7[0]);

eval-poly_deg128 <N,L,DNUM,K> (q, P, c+128, Delta, T1,T2,T4,T8,T16,T32,T64,
                               evk_hat, L7[1]);

eval-poly-merge <N,L-7,DNUM,K> (q, P, Delta, T128, evk_hat,
                                L7[0], L7[1], res_hat );

}
```