NTT Learning Notes

by github:Hxohu

在基于格构建的后量子密码体制中，要多次使用环上多项式乘法。但多项式乘法的时间复杂度为 $O(n^2)$，这将降低密码算法的运行效率。而通过 NTT 变换，可以将环上多项式乘法的时间复杂度将为 $O(n\log n)$，有效提升整体算法运行效率。在第一部分，将介绍多项式的两种表示方法及对应的乘法; 在第二部分，将介绍 NTT 方法的基本思想与方法; 在第三部分，将介绍在 $\mathbb{Z}_q/(x^n-1)$ 与 $\mathbb{Z}_q/(x^n+1)$ 两个常用多项式环上的乘法。在第四部分，将给出具体的实例来说明以上方法。

1 多项式的两种表示方法及对应的乘法

1.1 多项式的一般表示及对应乘法

一般情况下，将 n 次多项式定义如下：

$$f(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n=\sum_{i=0}^{n}a_ix^i$$

其中第 n 项系数 $a_n\neq 0$。

对于两个多项式 $f(x)=\sum_{i=0}^{n}a_ix^i$，$g(x)=\sum_{i=0}^{m}b_ix^i$：其乘法为：

$$h(x)=f(x)g(x)=\sum_{k=0}^{m+n}\left(\sum_{i+j=k}a_ib_j\right)x^k \tag{1}$$

即两多项式相乘后 x^k 的系数为 $f(x)$ 与 $g(x)$ 下标和为 k 对应项的系数乘积的和。由于其中含有两个求和符号，所以使用该表达式进行多项式乘法时间复杂度为 $O(n^2)$。

1.2 多项式的点值表示法及有关乘法

根据插值理论，如果有 $n+1$ 个点，则可以唯一恢复出一个 n 次多项式，即可以用 $n+1$ 个点来表示一个多项式。例如，可以使用点 $(-1,-3)$,$(0,2)$ 来表示一次多项式 $y=2+5x$。

对于一般情况，若 n 次多项式 $f(x)$, 已知 $n+1$ 个点:

$$(x_0,f(x_0)),(x_1,f(x_1)),\ldots,(x_n,f(x_n))$$

将其写为线性方程组的形式，即

$$\begin{cases}a_0+a_1x_0+a_2x_0^2+\cdots+a_nx_0^n=f(x_0)\\ a_0+a_1x_1+a_2x_1^2+\cdots+a_nx_1^n=f(x_1)\\ \qquad\qquad\qquad\qquad\vdots\\ a_0+a_1x_n+a_2x_n^2+\cdots+a_nx_n^n=f(x_n)\end{cases}$$

可将其进一步转化为矩阵表示：

$$\begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{2}$$

因此，在已知 $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$ 后可通过求解线性方程组来快速确定多项式系数 $a_0, a_1, \dots, a_n$。其中的 $n \times n$ 矩阵为范德蒙矩阵，设为 A。

例如，对于 $f(x) = 3x^2 + 2x - 2, g(x) = 2x^2 + 8$, 若求得如下 5 个值：

x	2	4	8	16	32
$f(x)$	14	54	206	798	3134
$g(x)$	16	40	136	520	2056

将对应纵坐标的点相乘得：

x	2	4	8	16	32
$f(x)g(x)$	224	2160	28016	414960	6443504

在得到这 5 个点后，通过求解线性方程组即可恢复 $f(x)g(x)$ 的系数并得到两多项式相乘的结果: $f(x)g(x) = 6x^4 + 4x^3 + 20x^2 + 16x - 16$。

在上述求解过程中，如果利用高斯消元法来求解对应的系数，时间复杂度为 $O(n^3)$，比直接相乘的时间复杂度更高。但是上述过程并没有对多项式及相关参数进行限制，而在格密码中如果能很好地控制多项式环的参数，就能有效地降低多项式乘法的时间复杂度。

2 NTT 方法的基本思想

NTT 的基本思想与多项式的点值表示的多项式乘法类似，即通过将多项式转化为不同的点值，再根据这些不同的点值的相乘来得到所需多项式的不同的点值，最后根据这些点值来逆向求解多项式乘法。在该过程中，将多项式转化为点值的过程称为 NTT(正向 NTT)，将点值恢复为多项式的过程称为 INTT(逆 NTT)。

在 NTT 中，可以通过参数的选取使选点采样的过程变为递归问题，使时间复杂度降为 $O(n \log n)$。

2.1 NTT 参数选取限制

在实际应用中，多项式次数多为偶数，因此以下均假设多项式次数 n 为偶数。

假设多项式 $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$，对所有奇数项提取 x 得：

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \\ &= (a_0 + a_2 x^2 + \cdots + a_n x^n) + x(a_1 + a_3 x^2 + \cdots + a_{n-1} x^{n-2}). \\ &= f_{even}(x) + x f_{odd}(x) \end{aligned} \tag{3}$$

其中 $f_{even}(x)$ 表示 $f(x)$ 中的偶数项,$f_{odd}(x)$ 表示 $f(x)$ 中的奇数项。若对 x 取反为 $-x$ 得：

$$f(-x) = f_{even}(x) - xf_{odd}(x) \tag{4}$$

公式 (3)(4) 为 CT 蝴蝶变换的基础，将在之后详细介绍。

取 $x_1 = x^2$ 得：

$$f(x_1) = (a_0 + a_2x_1 + \cdots + a_nx_1^{\frac{n}{2}}) + x_1(a_1 + a_3x_1 + \cdots + a_{n-1}x_1^{\frac{n-2}{2}})$$
$$f(-x_1) = (a_0 + a_2x_1 + \cdots + a_nx_1^{\frac{n}{2}}) - x_1(a_1 + a_3x_1 + \cdots + a_{n-1}x_1^{\frac{n-2}{2}})$$

该过程就是将原来求解 n 次多项式的问题转化为求解 $n/2$ 次多项式，并可以一直向下分解直至转化为求解一次多项式, 此过程即可用递归完成求解，并使时间复杂度降为 $O(n\log n)$。

但在该递归过程中，需要 $x_1 = x^2, x_2 = x_1^2, \ldots, 1 = x_{\frac{n}{2}-1}^2$，即 $x_{\frac{n}{2}-1}^2 = x^n = 1$。因此，在该过程中，首先需要找到一个 ω 使得 $\omega^n = 1$，再依次采样 $x_0 = \omega^0, x_1 = \omega^1, \ldots, x_n = \omega^{n-1}$。最后公式 (1) 中的范德蒙矩阵可写为:

$$A = \begin{bmatrix} \omega^{0\times0} & \omega^{1\times0} & \omega^{2\times0} & \ldots & \omega^{n\times0} \\ \omega^{0\times1} & \omega^{1\times1} & \omega^{2\times1} & \ldots & \omega^{n\times1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^{0\times n} & \omega^{1\times n} & \omega^{2\times n} & \ldots & \omega^{n\times n} \end{bmatrix}$$

此时，若 $\omega^0, \omega^1, \ldots, \omega^n$ 有任意一对或多对相等则会导致该矩阵行列式为 0，无逆矩阵可进行 INTT 变换。因此要求 $\omega^0, \omega^1, \ldots, \omega^n$ 两两不等。

因此，在使用 NTT 时需存在参数 ω，并有两点参数限制:

1.$\omega^n = 1$(保证递归可以进行)

2.$\omega^0, \omega^1, \ldots, \omega^n$ 两两不等 (保证 INTT 可以进行)

2.2 NTT 与 INTT 的定义

定义 NTT:

设 f 为 n 次多项式, 系数 $\boldsymbol{f} = (f_0, f_1, \ldots, f_n)$,$NTT(\boldsymbol{f}) = \hat{\boldsymbol{f}} = (\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_n)$, 其中:

$$\hat{f}_j = \sum_{i=0}^{n} \omega^{i\times j} f_i$$

该定义是对上述文本的总结，即 NTT 变换是在已知多项式系数的情况下将多项式变换为点值。而 INTT 变换则是在已知多项式的点值的情况下求出多项式的系数。

根据公式 (2) 可知，根据多项式的点值求出多项式系数，将该式同时左乘 A^{-1} 即可

得到多项式系数。通过对范德蒙矩阵求逆可得:

$$A^{-1} = n^{-1} \begin{bmatrix} \omega^{-0\times0} & \omega^{-0\times1} & \omega^{-0\times2} & \dots & \omega^{-0\times n} \\ \omega^{-1\times0} & \omega^{-1\times1} & \omega^{-1\times2} & \dots & \omega^{-1\times n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^{-n\times0} & \omega^{-n\times1} & \omega^{-n\times2} & \dots & \omega^{-n\times n} \end{bmatrix}$$

基于此，即可定义 INTT。

定义 INTT:

$INTT(\hat{\boldsymbol{f}}) = \boldsymbol{f} = (f_0, f_1, \dots, f_n)$, 其中：

$$f_i = n^{-1} \sum_{j=0}^{n} \omega^{-i\times j} \hat{f}_j$$

要能够使用 NTT 与 INTT 变换来实现多项式乘法，依赖于两个重要的性质:

1.$f = INTT(NTT(f))$

2.$NTT(f) \circ NTT(g) = NTT(f \circ g)$

其中 $\circ$ 表示向量对应位置元素相乘，因此可通过 $INTT(NTT(f) \circ NTT(g))$ 计算多项式 f, g 相乘后的结果。

3 在两种常用多项式环上的 NTT

在格密码方案中，所基于的多项式环多为 $\mathbb{Z}_q/(x^n-1)$ 与 $\mathbb{Z}_q/(x^n+1)$。其中，基于 $\mathbb{Z}_q/(x^n-1)$ 的多项式乘法称之为循环卷积 (Cyclic convolution, CC) 或正闭包卷积 PWC(Postive wrapped convolution), 基于此的 NTT 称为 CC-based NTT；基于 $\mathbb{Z}_q/(x^n+1)$ 的多项式乘法称之为负闭包卷积 (Negative wrapped convolution), 基于此的 NTT 称为 NWC-based NTT。

根据基于的环的不同，NTT 变换将会有不同的定义，但其核心思想均第二部分相同。在该部分，将依次介绍基于 $\mathbb{Z}_q/(x^n-1)$ 与 $\mathbb{Z}_q/(x^n+1)$ 环上的乘法定义，NTT 与 INTT 的定义，CT 蝴蝶变换与 GS 蝴蝶变换。

3.1 基于 $\mathbb{Z}_q/(x^n-1)$ 环上的 NTT

在该环上的 NTT 参数需要满足 n 为 2 的次幂，且 $q \equiv 1 mod n$, 其中 q 为素数。

在基于 $\mathbb{Z}_q/(x^n-1)$ 的环中，$x^n - 1 = 0$，即 $x^n = 1$。因此在此多项式环上，多项式的次数不会高于 n，即多项式次数最高为 $n-1$。

取在该环上的两个多项式 $f(x), g(x)$:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_{n-1} x^{n-1} = \sum_{i=0}^{n-1} f_i x^i$$

$$g(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g_{n-1} x^{n-1} = \sum_{i=0}^{n-1} g_i x^i$$

由于 $x^n = 1$, 所以 $x^{k+n} = x^k$, 即在该环上的多项式乘法为：

$$h(x) = f(x)g(x) = \sum_{k=0}^{n-1} \left(\left(\sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{n-1} f_i g_{k+n-i} \right) modq \right) x^k \tag{5}$$

根据 2.1 中的描述，为保证递归过程能够进行 n 需要为 2 的方幂；且 n 与 q 之间满足 $q \equiv 1(modn)$ 以保证存在 ω, 使得 $\omega^n \equiv 1(modq)$，且不存在 $n' < n$, 使得 $\omega^{n'} \equiv 1(modq)$，即 n 为 ω 的阶，其中 q 为素数，以下为其证明。

证明：

若 ω 的阶为 n', 分以下两种情况讨论：

1. 若 $n \not\equiv 0 modn'$, 则存在 m, r 使得 $n = mn' + r, 0 < r < n'$。若 $\omega^{n'} \equiv 1(modq)$，则 $\omega^n \equiv \omega^{mn'+r} \equiv \omega^r \equiv 1 modq$。即存在 $r < n'$ 且 $\omega^r \equiv 1 modq$, 与 n' 为 ω 的阶矛盾。

2. 若 $n \equiv 0 \equiv modn'$，则存在 m 使得 $mn' = n$, 此处不妨假设 $m < n'$。则 $\omega^n \equiv \omega^{mn'} \equiv \omega^m \omega^{n'} \equiv 1 modq$ 根据假设 $\omega^{n'} \equiv 1 modq$, 则 $\omega^m \equiv 1 modq$, 与 ω 的阶为 n' 矛盾。(因 n 为 2 的次幂，则 n', m 同样将为 2 的次幂，若 $m > n'$ 则可继续向下划分，推出矛盾)

可以看到，式 (1) 与式 (5) 并无本质区别，因此基于 $\mathbb{Z}_q/(x^n - 1)$ 环上的 NTT 与未加限制的 NTT 与 INTT 定义并无不同，即:

$NTT(\boldsymbol{f}) = \hat{\boldsymbol{f}} = (\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_n)$, 其中 $\hat{f}_j = \sum_{i=0}^{n-1} \omega^{i \times j} f_i$

$INTT(\hat{\boldsymbol{f}}) = \boldsymbol{f} = (f_0, f_1, \ldots, f_n)$, 其中 $f_i = n^{-1} \sum_{j=0}^{n-1} \omega^{-i \times j} \hat{f}_j$

3.2 基于 $\mathbb{Z}_q/(x^n + 1)$ 环上的 NTT

在该环上的 NTT 参数需要满足 n 为 2 的次幂，且 $q \equiv 1 mod 2n$, 其中 q 为素数。

在基于 $\mathbb{Z}_q/(x^n + 1)$ 的环中，其多项式次数最高为 $n - 1$, 但在该环中 $x^n + 1 = 0$，即 $x^n = -1$，所以 $x^{k+n} = -x^k$。因此该环上的多项式乘法为:

$$h(x) = f(x)g(x) = \sum_{k=0}^{n-1} \left(\left(\sum_{i=0}^{k} f_i g_{k-i} - \sum_{i=k+1}^{n-1} f_i g_{k+n-i} \right) modq \right) x^k \tag{6}$$

在这个环上的多项式乘法与 $\mathbb{Z}_q/(x^n - 1)$ 环上的多项式乘法相比，差异在于内层括号的符号不同。

根据条件 $q \equiv 1 mod 2n$，以及 3.1 节中的相关内容，在环 $\mathbb{Z}_q/(x^n + 1)$ 中必然存在 $\psi, \omega, s.t. \psi^n \equiv -1 modq, \omega^{2n} \equiv 1 modq$。

在 $\mathbb{Z}_q/(x^n - 1)$ 中，$x^{n+1} = x$; 而在 $\mathbb{Z}_q/(x^n + 1)$ 中 $x^{n+1} = -x$，但是对于 ψx 则有 $(\psi x)^{n+1} = \psi^{n+1} x^{n+1} = (-\psi)(-x) = \psi x$。因此，若将 $\mathbb{Z}_q/(x^n + 1)$ 的元素 x 映射到 $\mathbb{Z}_q/(x^n - 1)$ 上，则需要多乘一个 ψ，使得 x 变为 ψx。通过上述映射，则公式 (6) 则

可以写为：

$$
\begin{aligned}
h(x) &= f(x)g(x) \\
&= \sum_{k=0}^{n-1}\left(\left(\sum_{i=0}^{k}\psi^i f_i\cdot\psi^{k-i}g_{k-i}-\sum_{i=k+1}^{n-1}\psi^i f_i\cdot\psi^{k+n-i}g_{k+n-i}\right)modq\right)x^k \\
&= \sum_{k=0}^{n-1}\left(\left(\sum_{i=0}^{k}\psi^k f_i g_{k-i}-\sum_{i=k+1}^{n-1}\psi^{k+n} f_i g_{k+n-i}\right)modq\right)x^k \\
&= \sum_{k=0}^{n-1}\left(\left(\sum_{i=0}^{k}\psi^k f_i g_{k-i}+\psi^k\sum_{i=k+1}^{n-1} f_i g_{k+n-i}\right)modq\right)x^k
\end{aligned}
\tag{7}
$$

因此，在 $\mathbb{Z}_q/(x^n+1)$ 中的 NTT 与 INTT 定义为:

$NTT^{\psi}(\boldsymbol{f})=\hat{\boldsymbol{f}}=(\hat{f}_0,\hat{f}_1,\ldots,\hat{f}_n)$, 其中 $\hat{f}_j=\sum_{i=0}^{n-1}\psi^i\omega^{i\times j}f_i$

$INTT^{\psi}(\hat{\boldsymbol{f}})=\boldsymbol{f}=(f_0,f_1,\ldots,f_n)$, 其中 $f_i=n^{-1}\sum_{j=0}^{n-1}\psi^{-j}\omega^{-i\times j}\hat{f}_j$

由上述结论知 $\omega\equiv\psi^2\equiv 1 modq$, 因此 NTT 与 INTT 定义可进一步化简为:

$NTT^{\psi}(\boldsymbol{f})=\hat{\boldsymbol{f}}=(\hat{f}_0,\hat{f}_1,\ldots,\hat{f}_n)$, 其中 $\hat{f}_j=\sum_{i=0}^{n-1}\psi^{2ij+i}f_i$

$INTT^{\psi}(\hat{\boldsymbol{f}})=\boldsymbol{f}=(f_0,f_1,\ldots,f_n)$, 其中 $f_i=n^{-1}\sum_{j=0}^{n-1}\psi^{-(2ij+j)}\hat{f}_j$

此处给出另一种理解 $\mathbb{Z}_q/(x^n+1)$ 中的 NTT 的角度。对于多项式 $x^{2n}-1$ 可以被分解为 $(x^n-1)(x^n+1)$，那么在环 $\mathbb{Z}_q/(x^n+1)$ 中的 NTT 应该是环 $\mathbb{Z}_q/(x^{2n}-1)$ 的一部分，因此环 $\mathbb{Z}_q/(x^n+1)$ 中的参数要求都与 $2n$ 相关。

3.3 CT 蝴蝶变换与 GS 蝴蝶变换

CT 蝴蝶变换是将 NTT 变换不断按照奇偶划分，将递归过程具象化，并达到加速的目的；而 GS 蝴蝶变换是 CT 蝴蝶变换的逆变换，并且加速 INTT 变换。待参数确定后，CT 蝴蝶变换与 GS 蝴蝶变换的各个过程也随之确定，即可将递归过程具象为带有 ω 或 ψ 不同幂次的表达式，通过预计算确定 ω 或 ψ 不同幂次就可直接代入多项式系数进行计算，有效提升运算效率。

在不同环上的 CT 蝴蝶变换与 GS 蝴蝶变换并没有太大区别，因此以下介绍均基于 $\mathbb{Z}_q/(x^n+1)$ 环。

在 $\mathbb{Z}_q/(x^n+1)$ 中，存在 $\psi^n\equiv -1 modq, \psi^{2n}\equiv 1 modq$。基于此，若对 NTT 变换

按照奇偶划分得:

$$\begin{aligned}\hat{f}_j &= \sum_{i=0}^{n-1} \psi^{2ij+i} f_i \quad modq \\ &= \sum_{i=0}^{n/2-1} \psi^{4ij+2i} f_{2i} + \sum_{i=0}^{n/2-1} \psi^{4ij+2i+2j+1} f_{2i+1} \quad modq \\ &= \sum_{i=0}^{n/2-1} \psi^{4ij+2i} f_{2i} + \psi^{2j+1} \sum_{i=0}^{n/2-1} \psi^{4ij+2i} f_{2i+1} \quad modq\end{aligned} \tag{8}$$

将 j 代换为 $j+n/2$, 则 $\psi^{(j+n/2)+1} = \psi^{2j+n+1} = -\psi^{2j+1}$，因此：

$$\hat{f}_{j+n/2} = \sum_{i=0}^{n/2-1} \psi^{4ij+2i} f_{2i} - \psi^{2j+1} \sum_{i=0}^{n/2-1} \psi^{4ij+2i} f_{2i+1} \quad modq \tag{9}$$

设 $A_j = \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i}, B_j = \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i+1}$, 有：

$$\hat{f}_j = A_j + \psi^{2j+1} B_j \quad modq \tag{10}$$

$$\hat{f}_{j+n/2} = A_j - \psi^{2j+1} B_j \quad modq \tag{11}$$

因此，通过重复利用 A_j, B_j 的值计算得到 $\hat{f}_j, \hat{f}_{j+n/2}$, 并且 A_j, B_j 的值也可以通过下一层的计算得到。该过程可表示为下图:

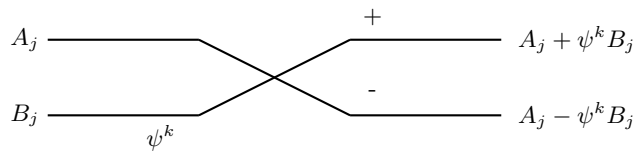

图 1: CT 蝴蝶变换

GS 蝴蝶变换是 CT 蝴蝶变换的逆变换, 其过程可参照下图：

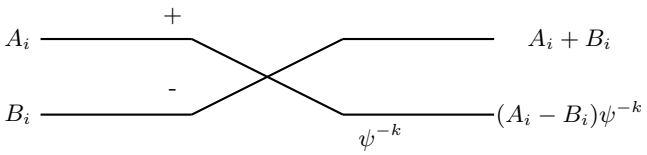

图 2: GS 蝴蝶变换

将 CT 蝴蝶变换的输出作为 GS 蝴蝶变换的输入即可恢复原来的多项式。在下一部分，将给出具体的实例来描述 NTT，INTT 变换与两种蝴蝶变换。

4 实例

在该部分，将基于 $\mathbb{Z}_q/(x^n+1)$ 环，选取 $n=4, q=7681, \psi=1925$ 来说明 NTT、INTT 变换与两种蝴蝶变换。

选取 $f(x)=1+2x+3x^2+4x^3, g(x)=5+6x+7x^2+8x^3$, 根据定义可知相乘后结果为：

$$h(x)=f(x)g(x)=-56-36x+2x^2+60x^3 \tag{12}$$

在此之前，首先给出比特翻转的定义，以便后续说明:

定义　比特翻转:

若 n 为 2 的次幂，b 为非负数且 $n<n$, 则 b 比特翻转后为:

$$brv_n(b_{log(n-1)2^{log(n-1)}}+\cdots+b_1 2+b_0)=b_0 2^{log(n-1)}+\cdots+b_{log(n-2)}2+b_{log(n-1)}$$

例如，若 $n=256, b=69$，则 $b=(0100\quad 0101)_2$,b 比特翻转后为 $brv_{256}(b)=(1010\quad 0010)_2$。

对 $\boldsymbol{f}$ 各系数使用数组存储，即 $\boldsymbol{f}=(1,2,3,4)$, 设经过 CT 蝴蝶变换的结果为 $\hat{\boldsymbol{f}}=(\hat{f}[0],\hat{f}[1],\hat{f}[2],\hat{f}[3])$ 具体过程可参照下图:

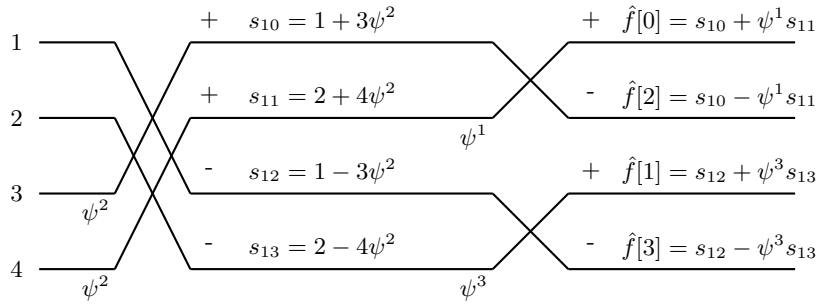

图 3: 通过 CT 蝴蝶变换求 $NTT(\boldsymbol{f})$

根据图 3, 并使用 $\psi^4\equiv -1 mod q$ 进行代换知:

$$\begin{aligned}
\hat{f}[0]&=\psi^0(1+3\psi^2)+\psi^1(2+4\psi^2)=1\psi^0+2\psi^1+3\psi^2+4\psi^3\\
\hat{f}[1]&=\psi^0(1-3\psi^2)+\psi^3(2-4\psi^2)=1\psi^0+2\psi^3-3\psi^2+4\psi^1\\
\hat{f}[2]&=\psi^0(1+3\psi^2)-\psi^1(2+4\psi^2)=1\psi^0-2\psi^1+3\psi^2-4\psi^3\\
\hat{f}[3]&=\psi^0(1-3\psi^2)-\psi^3(2-4\psi^2)=1\psi^0-2\psi^1-3\psi^2+4\psi^1
\end{aligned}$$

可以看到，此结果与 NTT 定义所得到的结果相同。但是由于 CT 蝴蝶变换的输出会导致下标比特翻转，因此 CT 蝴蝶变换的输出为 $\hat{\boldsymbol{f}}'=[1467,3471,2807,7621]$，而 $\hat{\boldsymbol{f}}=[1467,2807,3471,7621]$。

在该过程中，需要进行 $logn$ 层，且每一层需要计算 n 次。因此，该算法的时间复杂度为 $O(nlogn)$。

此处给出对 $f(x)$ 进行 CT 蝴蝶变换时的各中间值:

$$\psi^1 = 1925, \psi^2 = 3383, \psi^3 = 6468, s_{10} = 2469, s_{11} = 5853, s_{12} = 5214, s_{13} = 1832$$

将 CT 蝴蝶变换的输出作为 GS 蝴蝶变换的输入，具体可参见下图：

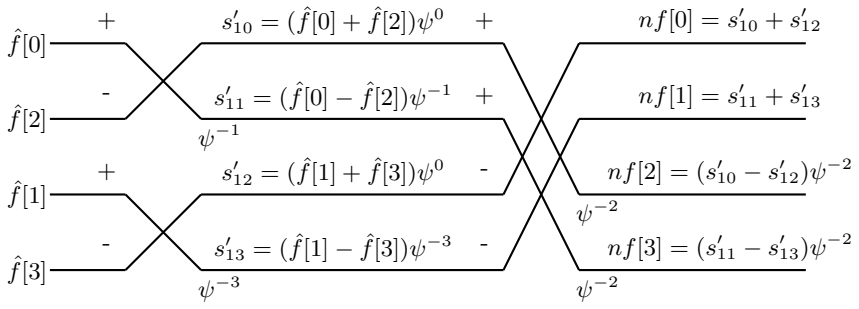

图 4: 通过 GS 蝴蝶变换求 $INTT(\hat{\boldsymbol{f}})$

根据图 4，并使用 $\psi^4 \equiv \psi^{-4} \equiv -1 mod q$ 进行代换知:

$$\begin{aligned}
nf[0] &= ((\hat{f}[0] + \hat{f}[2])\psi^{-0} + (\hat{f}[1] + \hat{f}[3])\psi^{-0})\psi^{-0} \\
&= \hat{f}[0]\psi^{-0} + \hat{f}[1]\psi^{-0} + \hat{f}[2]\psi^{-0} + \hat{f}[3]\psi^{-0} \\
nf[1] &= ((\hat{f}[0] - \hat{f}[2])\psi^{-1} + (\hat{f}[1] - \hat{f}[3])\psi^{-3})\psi^{-0} \\
&= \hat{f}[0]\psi^{-1} + \hat{f}[1]\psi^{-3} - \hat{f}[2]\psi^{-1} - \hat{f}[3]\psi^{-3} \\
nf[2] &= ((\hat{f}[0] + \hat{f}[2])\psi^{-0} - (\hat{f}[1] + \hat{f}[3])\psi^{-0})\psi^{-2} \\
&= \hat{f}[0]\psi^{-2} - \hat{f}[1]\psi^{-2} + \hat{f}[2]\psi^{-2} - \hat{f}[3]\psi^{-2} \\
nf[3] &= ((\hat{f}[0] - \hat{f}[2])\psi^{-1} - (\hat{f}[1] - \hat{f}[3])\psi^{-3})\psi^{-2} \\
&= \hat{f}[0]\psi^{-3} + \hat{f}[1]\psi^{-1} - \hat{f}[2]\psi^{-3} - \hat{f}[3]\psi^{-1}
\end{aligned}$$

可以看到，此结果与 INTT 定义所得到的结果相同，且按照比特翻转后的顺序输入后得到的顺序为正常顺序。此过程时间复杂度仍为 $O(nlogn)$。

对得到的结果乘以 n^{-1} 即可得到：

$$f[0] = 4 \times 4^{-1} = 1, f[1] = 8 \times 4^{-1} = 2, f[2] = 12 \times 4^{-1} = 3, f[3] = 16 \times 4^{-1} = 4$$

此处给出对 $\hat{\boldsymbol{f}}$ 进行 GS 蝴蝶变换的各中间值:

$$\psi^{-1} = 1213, \psi^{-2} = 4298, \psi^{-3} = 5756, s'_{10} = 4938, s'_{11} = 4025, s'_{12} = 2747, s'_{13} = 3664$$

以下给出使用 NTT 计算 $h(x) = f(x)g(x)$ 的过程:

1. 计算 $\hat{\boldsymbol{f}} = NTT(\boldsymbol{f}) = (1467, 2087, 3471, 7621)$,
   $\hat{\boldsymbol{g}} = NTT(\boldsymbol{g}) = (2489, 7489, 6478, 6607)$;
2. 计算 $\hat{\boldsymbol{h}} = \hat{\boldsymbol{f}} \circ \hat{\boldsymbol{g}} = (2888, 6407, 2851, 2992)$;
3. 计算 $\boldsymbol{h} = INTT(\hat{\boldsymbol{h}}) = (7625, 7645, 2, 60) = (-56, -36, 2, 60)$。

以上显示的均为正常顺序，但在实际操作中可以直接让经过 NTT 变换后的两多项式相乘，并按比特翻转的顺序输入 INTT，最终仍将按正常顺序输出两多项式相乘后的结果。而 NTT 变换与 INTT 变换的时间复杂度均为 $O(nlogn)$, 因此通过此方法即可将多项式在乘法的时间复杂度由 $O(n^2)$ 降低为 $O(nlogn)$。

5 结语

本文对 NTT 变换的思想与过程进行了说明，希望能够通过此对 NTT 有一个初步认识，因此很多结论的证明并未给出，主要参考的资料有 [1] [2] [3]。在理论上，有“上裁法”与“下裁法”来使 NTT 的参数限制变得更为宽松，也可综合两种方法达到该效果，即“混合 NTT”[4]; 并且可以综合使用 Karatsuba 等技巧 [5]，使 NTT 的速度变得更快。在实现上，则有对基于 AVX2，ARM Cortex-M4，GPU 等的 NTT 多平台实现的研究，力求使 NTT 能够在各种软硬件上更快实现。也有基于 NTT 的特性，分析其在硬件实现上的能耗等特性从而对基于格的密码体制进行侧信道分析的研究 [6]。可以根据这些方向，来进一步学习 NTT。

参考文献

[1] 微信公众号：Coder 小 Q. 格密码学习笔记 (十八) 乘法运算加速: 快速数论变换, 2024.

[2] Ardianto Satriawan, Infall Syafalni, Rella Mareta, Isa Anshori, Wervyan Shalannanda, and Aleams Barra. Conceptual review on number theoretic transform and comprehensive review on its implementations. *IEEE Access*, 11:70288–70316, 2023.

[3] Zhichuang Liang and Yunlei Zhao. Number theoretic transform and its applications in lattice-based cryptosystems: A survey. *arXiv preprint arXiv:2211.13546*, 2022.

[4] Shuai Zhou, Haiyang Xue, Daode Zhang, Kunpeng Wang, Xianhui Lu, Bao Li, and Jingnan He. Preprocess-then-ntt technique and its applications to k yber and n ew h ope. In *Information Security and Cryptology: 14th International Conference, Inscrypt 2018, Fuzhou, China, December 14-17, 2018, Revised Selected Papers 14*, pages 117–137. Springer, 2019.

[5] Yiming Zhu, Zhen Liu, and Yanbin Pan. When ntt meets karatsuba: preprocess-then-ntt technique revisited. In *International conference on information and communications security*, pages 249–264. Springer, 2021.

[6] Tianrun Yu, Chi Cheng, Zilong Yang, Yingchen Wang, Yanbin Pan, and Jian Weng. Hints from hertz: Dynamic frequency scaling side-channel analysis of number theoretic transform in lattice-based kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):200–223, 2024.