南開大學
Nankai University

计算机学院

并行程序设计实验报告

基于全同态加密的并行优化设计

姓名：蒋佳豪 | 孔德嵘

学号：2211103 | 2213626

专业：计算机科学与技术

2024 年 4 月 7 日

目录

1 问题描述

1.1 引入

全同态加密（Fully Homomorphic Encryption，FHE）是一项前沿的密码学技术，允许在加密状态下进行计算，而无需解密数据。FHE 目前的实现仍然面临一些性能和效率上的挑战，但全同态加密作为一项前沿技术，正逐渐成为数据安全和隐私保护领域的重要工具。本文将围绕并行优化同态加密性能展开，旨在提升全同态加密中编码理论、密文运算的性能。以“加速大概率事件”的设计理念，并行优化 NTT、INTT, 并用 Packing 的编码理论，在密文中实现 SIMD 的操作，加快全同态加密算法的性能。

目前的工程 fork 于 HEHUB，数据集和日后的代码优化和工程更新，详见此仓库

1.2 FFT 和 NTT 的理论证明

1.2.1 前置数论知识

1. 多项式

   多项式的简单定义：对于求和式 $\sum a_n x^n$，如果是有限项相加，称为多项式，，记作 $f(x) = \sum_{n=0}^{m} a_n x^n$。

   更严谨的定义：给定一个环 R(通常为交换环) 及一个未知数 x，则任何形同

   $$a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1} + a_n x^n$$

   的的代数表达式叫做 R 上的一元多项式，简称为多项式。

   对于多项式操作，最重要的就是多项式乘法。给定多项式 $f(x)$ 和 $g(x)$：

   $$f(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1} + a_n x^n \tag{1}$$
   $$g(x) = b_0 + b_1 x + b_2 x^2 + ... + b_{m-1} x^{m-1} + b_m x^m \tag{2}$$

   定义两者相乘 $h(x) = f(x) \cdot g(x)$：

   $$h(x) = \sum_{i=0}^{n} \sum_{j=0}^{m} a_i b_j x^{i+j} = c_0 + c_1 x + ... + c_{n+m} x^{n+m}$$

   多项式的表示：多项式有两种表示方法，系数表示法和点值表示法。给出 x 各项的系数为系数表示法，例如上文的 $f(x)$、$g(x)$、$h(x)$ 均为系数表示法，系数表示法也是最为常见的多项式表示法。给出 n 组满足多项式的互不相同的坐标点值，可以唯一确定多项式的形式和系数，这就是点值表示法，下面介绍点值表示法。

   对于任意多项式 $f(x)$，如果代入 $(n+1)$ 个不同点，必得到 $(n+1)$ 个不同的值，即 $(n+1)$ 个横坐标纵坐标均无重复的坐标点值。换成向量的语言解释，取 $n+1$ 个不同的 x 组成向量 $X = [x_0, x_1, ..., x_n]$，分别代入多项式，可得 $n+1$ 个 y，记为向量 $Y = [y_0, y_1, ..., y_n]$，代入过程

可转化为矩阵乘法的形式：

$$\begin{bmatrix} y_0 \\ y_1 \\ ... \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & ... & x_0^n \\ 1 & x_1 & x_1^2 & ... & x_1^n \\ ... & ... & ... & ... & ... \\ 1 & x_n & x_n^2 & ... & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ ... \\ a_n \end{bmatrix}$$

展开成非齐次线性方程组：

$$y_0 = a_0 + a_1x_0 + a_2x_0^2 + ... + a_{n-1}x_0^{n-1} + a_nx_0^n \tag{3}$$

$$y_1 = a_0 + a_1x_1 + a_2x_1^2 + ... + a_{n-1}x_1^{n-1} + a_nx_1^n \tag{4}$$

$$... \tag{5}$$

$$y_n = a_0 + a_1x_n + a_2x_n^2 + ... + a_{n-1}x_n^{n-1} + a_nx_n^n \tag{6}$$

计 $n+1$ 方阵为 D，非齐次线性方程组有无穷解当且仅当系数阵 D 的秩与增广矩阵 $D|A^T$ 的秩相等，满秩时有唯一解。容易发现 D^T 的行列式是范德蒙德行列式，所以

$$|D| = |D^T| = \prod_{n \geq i > j \geq 0} (x_i - x_j)$$

由于 x_i 互不相同，可得 $|D| > 0$，即系数阵 D 满秩，且系数阵 D 的秩与增广矩阵 $D|A^T$ 的秩相等，所以可由 n 个不同点对唯一确定多项式的参数。

同时由于系矩阵 D 满秩，所以存在 D^{-1}，若把系矩阵 D 理解为一个域内的变换，则 D^{-1} 为其逆变换。即已知向量 A 可求向量 Y，已知向量 Y 可求向量 A。而对于多项式来说，向量 A 代表系数表示法，向量 Y 代表在给定向量 X 的前提下的点值表示法，所以如果能找到一个变换，其矩阵的行列式表示是范德蒙德行列式，即可完成点值转系数的双向转换。

下面考虑如何求解多项式乘法 $h(x)$。

已知 $f(x)$ 和 $g(x)$ 的系数表示法求多项式乘法的算法显而易见，这里不多作介绍，时间复杂度是 $O(n^2)$, 且很难进行优化，但如果已知两个多项式的点值表示法怎样求多项式乘法呢?

考虑式子 $h(x) = f(x) \cdot g(x)$，$f(x)$ 次数为 n，点值对为 $n+1$，$g(x)$ 次数为 m，点值对为 $m+1$，则 $h(x)$ 次数为 $n+m$，点值对为 $n+m+1$。如果用点值表示法对多项式进行加减乘的运算，假设 $n < m$，则必须保证 $A = \{x|\forall x, s.t.(x,y) \in f(x))\} \subseteq B = \{x|\forall x, s.t.(x,y) \in g(x))\}$。由于在同一个 x 的条件下，$f(x) \cdot g(x)$ 本身表示的就是两个多项式的值相乘。所以使用点值表示法进行多项式相乘时，对于 $x \in A \cap B$，直接将对应纵坐标相乘可得到 n 个 $h(x)$，对于 $x \in B - A$，纵坐标为 0，但是由于 $h(x)$ 需要 $n+m+1$ 个点值确认，所以 $f(x)$ 和 $g(x)$ 共需要 $n+m+1$ 个点值。

因此如果已知了两个多项式的点值表示法，求两多项式相乘结果的点值表示法，时间复杂度是 $O(n)$ 的，但是一个多项式从系数表示法转换为点值表示法使用暴力算法依次代入，即构造普通系数矩阵, 时间复杂度是 $O(n^2)$。如果我们能构造一个正确的系数矩阵 (变换), 实现以低于 $O(n^2)$ 的时间复杂度对多项式的系数表示法和点值表示法进行转换，那么多项式相乘就可以优化到低于 $O(n^2)$ 的时间复杂度。

2. 复数

介绍复数的目的是引出单位根，方便 FFT 的计算，因此忽略一些高中基础定义。

每一个复数 $a+bi$ 都可以在坐标轴上表示出来，复数 $z=r(cos\theta+isin\theta)$ 表示点 $(rcos\theta, rsin\theta)$，同时从圆和角的角度考虑，又表示以 r 为半径的圆上辐角为 θ 的点。

由欧拉公式可将复数做一下变换，方便计算：

$$e^{i\theta}=cos\theta+isin\theta$$

复数相乘也有几何意义，取复数 $z_1=a+bi=r_1(cos\theta_1+isin\theta_1), z_2=c+di=r_2(cos\theta_2+isin\theta_2)$，定义

$$z_1z_2=(ac-bd)+(bc+ad)i$$

$$=r_1r_2(cos\theta_1+\theta_2+isin\theta_1+\theta_2)$$

其模长等于两个复数的模长相乘，辐角等于两个复数的辐角相加，几何意义表示 z_2 进行了 z_1 的模长伸缩和辐角旋转。

在复数域下，满足 $x^n=1$ 的 x 称为 n 次单位根，共有 n 个解，将所有的 n 次单位根按照辐角大小排列，第 k 个 n 次单位根记作 ω_n^k，ω_n^1 记为本原单位根，几何意义上，ω_n^k 可看作 k 个本原单位根的辐角进行叠加，，代数上，0 到 $n-1$ 次方的值能生成所有 n 次单位根的 n 次单位根称为 n 次本原单位根。同时，按照欧拉公式的写法 $\omega_n^k=e^{i\frac{2k\pi}{n}}$。

代数上，这 n 个解构成了阿贝尔群，几何上，这 n 个不同的解在复平面上平分单位圆，他们与单位圆的 n 等分线所在线段重合。

接下来有三个比较重要的公式：

$$\omega_n^n=1$$

$$\omega_n^k=\omega_{2n}^{2k}$$

$$\omega_{2n}^k=-\omega_{2n}^{k+n}$$

第一个公式显而易见，下面分别从几何和代数两个方面证明后两个公式。

几何上，第一个公式表示将本原单位根的辐角分成了一半，把增加的个数翻倍，前后不变。第二个公式，ω_{2n}^{k+n} 代表 ω_{2n}^k 旋转过半个单位圆，两点原点对称。

代数上，第一个公式：

$$\omega_{2n}^{2k}=e^{i\frac{2\times2k\pi}{2n}}=e^{i\frac{2\times k\pi}{n}}=e^{i\frac{2k\pi}{n}}$$

第二个公式：

$$\omega_{2n}^{k+n}=e^{i\frac{2(k+n)\pi}{2n}}=e^{i(\frac{2k\pi}{2n}+\frac{\pi}{2})}=-e^{i\frac{2k\pi}{2n}}=\omega_{2n}^k$$

3. 原根

在了解原根相关概念前，需要一点数论的知识，包括欧拉定理、费马小定理、乘法逆元等相关概念。

欧拉定理：若 $gcd(a,m)=1$，则 $a^{\phi(m)} \equiv 1\ (mod\ m)$，其中 $\phi(m)$ 表示 m 的欧拉函数。

乘法逆元：若线性同余方程 $ax \equiv 1\ (mod\ b)$，则 x 称称为 $a\ mod\ b$ 的逆元，记作 a^{-1}，乘法逆元一般用来处理带有除法的模运算，求逆元有快速幂和线性两种方式。

下面介绍原根，阶的定义：若 $gcd(a,m)=1$，那么我们把满足 $a^k \equiv 1\ (mod\ m)$ 的最小正整数 k 称为 a 的模 m 的阶。例如 $5^2 \equiv 1\ (mod\ 3)$ 表明 5 的模 3 的阶是 2。

推理：如果 $a^n \equiv 1\ (mod\ m)$，那么 a 的阶整除 n，证明略。由欧拉定律知，若 $gcd(a,m)=1$，$a^{\phi(m)} \equiv 1\ (mod\ m)$，因此 a 的模 m 的阶一定是 $\phi(m)$ 的因数，我们把阶等于 $\phi(m)$ 的余数 (同余类) 称为模 m 的原根。下面以 2 6 为例。

$m=2$，可逆元有 1 共 $\phi(2)=1$ 个，其阶为 1，所以 1 是模 2 的原根。

$m=3$，可逆元有 1,2 共 $\phi(2)=2$ 个，2 阶为 2，所以 2 是模 3 的原根。

$m=4$，可逆元有 1,3 共 $\phi(2)=2$ 个，3 阶为 2，所以 3 是模 4 的原根。

$m=5$，可逆元有 1,2,3,4 共 $\phi(2)=4$ 个，2 阶为 4，3 阶为 4，4 阶为 2，所以 2,3 是模 5 的原根。

$m=6$，可逆元有 1,5 共 $\phi(2)=2$ 个，5 阶为 2，所以 2 是模 6 的原根。

原根存在定理：$m=1,2,4,p^\alpha,2p^\alpha$。

原根判定定理：若 $gcd(a,m)=1$，g 是 m 的原根当且仅当 $\phi(m)$ 的全部质因子 q_i 都满足 $g^{\frac{\phi(m)}{q_i}} \neq 1\ (mod\ m)$。

原根跟单位根有一些十分相似的性质（为了区分单位根和原根，原根里的 n 写作 N）

类似本原 n 次单位根，可以定义本原 n 次方根：在一个有限域内，只要 n 可除 $m-1$，就存在本原 n 次方根 g_N，且 $(g_N^k)^N \equiv 1\ (mod\ m)$，$g_N = g^{\frac{m-1}{N}}$。对于质数 $p=qN+1, n=2^q$，原根 g 满足 $g^{qN} \equiv 1\ (mod\ p)$。

原根还有以下性质

$$g_N^N \equiv 1\ (mod\ p)$$

$$g_{2N}^{2k} \equiv g_N^k (mod\ p)$$

$$g_N^{\frac{N}{2}} = g_N^{\frac{p-1}{2}} \equiv -1\ (mod\ p)$$

1.2.2 快速傅里叶变换 FFT

1. 引入傅里叶变换（Fourier Transform）是一种分析信号的方法，它可分析信号的成分，也可用这些成分合成信号。许多波形可作为信号的成分，傅里叶变换用正弦波作为信号的成分。设 $f(t)$ 是关于时间的函数，则傅里叶变换可以检测频率 ω 的周期在 $f(t)$ 出现的频率傅里叶变换符号为 $\mathbb{F}$，

$$F(\omega) = \mathbb{F}|f(t)| = \int_{-\infty}^{+\infty} f(t)e^{i\omega t}dt$$

逆变换：

$$f(t) = \mathbb{F}^{-1}|F(\omega)| = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F(\omega)e^{i\omega t}dw$$

分母 2π 恰好是指数函数的周期，傅里叶变换相当于将时域的函数与周期为 2π 的复指数函数进行连续的内积。逆变换仍旧为一个内积。

2. 离散傅里叶变换 (DFT) 离散傅里叶变换（Discrete Fourier transform，DFT）是傅里叶变换在时域和频域上都呈离散的形式，将信号的时域采样变换为频域采样。傅里叶变换是积分形式的连续的函数内积，离散傅里叶变换是求和形式的内积。

   设 $\{x_n\}_{n=0}^{N-1}$ 是某一满足有限性条件的序列，它的离散傅里叶变换（DFT）为：

$$X_k = \sum_{n=0}^{N} x_n e^{-i\frac{2k\pi}{N}n} = \sum_{n=0}^{N} x_n (\omega_N^k)^n$$

   ，记为：

$$\hat{x} = \mathbb{F}x$$

   逆变换（IDFT）：

$$x_n = \frac{1}{N}\sum_{n=0}^{N} X_k e^{i\frac{2k\pi}{N}n} = \frac{1}{N}\sum_{n=0}^{N} X_k (-\omega_N^k)^n$$

   记为：

$$x = \mathbb{F}^{-}1\hat{x}$$

   前置知识中提到，我们需要找到一个变换，其矩阵的行列式表示是范德蒙德行列式，可完成点值转系数的双向转换，傅里叶变换恰好就能满足这个特点，其矩阵表示为：

$$\begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega_n^1 & (\omega_n^1)^2 & ... & (\omega_n^1)^n \\ 1 & \omega_n^2 & (\omega_n^2)^2 & ... & (\omega_n^2)^n \\ ... & ... & ... & ... & ... \\ 1 & \omega_n^n & (\omega_n^n)^2 & ... & (\omega_n^n)^n \end{bmatrix}$$

   初始问题是 $Y = \mathbb{F}A$ 中，已知向量 A 求向量 Y。直接将矩阵和向量相乘是 n^2 的，下面考虑使用 FFT 进行优化。

3. 快速傅里叶变换 (FFT)

   设 $y = f(x), y_i = f(x_i)$，我们把 $f(x)$ 按照 x 系数角标的奇偶性进行分类：

$$f(x) = \sum_{i=0}^{n} a_i x^i = \sum_{i=0}^{n/2} a_{2i} x^{2i} + \sum_{i=0}^{n/2} a_{2i+1} x^{2i+1} = \sum_{i=0}^{n} a_{2i} x^{2i} + x\sum_{i=0}^{n-1} a_{2i+1} x^{2i}$$

   设

$$G(x) = \sum_{i=0}^{n/2} a_{2i} x^{2i}$$

$$H(x) = x\sum_{i=0}^{n/2} a_{2i+1} x^{2i}$$

   则

$$f(x) = G(x^2) + xH(x^2)$$

   把 $x = \omega_n^k$ 代入

$$f(\omega_n^k) = G((\omega_n^k)^2) + \omega_n^k H((\omega_n^k)^2)$$

利用单位根的性质进行化简

$$f(\omega_n^k) = G(\omega_n^{2k}) + \omega_n^k H(\omega_n^{2k}) = G(\omega_{\frac{n}{2}}^k) + \omega_n^k H(\omega_{\frac{n}{2}}^k)$$

把 $x = \omega_n^{k+\frac{n}{2}}$ 代入

$$f(\omega_n^{k+\frac{n}{2}}) = G(\omega_n^{2k+n}) + \omega_n^{2k+n} H(\omega_n^{2k+n})$$

化简

$$f(\omega_n^{k+\frac{n}{2}}) = G(\omega_n^{2k}) - \omega_n^k H(\omega_n^{2k}) = G(\omega_{\frac{n}{2}}^k) - \omega_n^k H(\omega_{\frac{n}{2}}^k)$$

结合两个化简后的式子可以发现，当求出 $G(\omega_n^{2k})$ 和 $H(\omega_n^{2k})$ 后，可以同时求出 $f(\omega_n^k)$ 和 $f(\omega_n^{k+\frac{n}{2}})$，可以通过归并的方式实现这种求解。

时间复杂度，考虑主定理：

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

由于 $O(n) = \Theta(n)$，因此

$$T(n) = \Theta(n^{log_2^2} log(n)) = \Theta(nlog(n))$$

4. 快速傅里叶逆变换 (IFFT)

通过 FFT，可以用 $O(nlog(n))$ 的快速傅里叶变换将多项式的系数表示法转化为点值表示法，现在还需要用快速傅里叶逆变换将多项式的系数表示法转化为点值表示法。

现在问题是 $Y = \mathbb{F}A$ 中，已知向量 Y 求向量 A。等号两边同时左乘系数矩阵的逆矩阵

$$\mathbb{F}^{-1}Y = A$$

求解范德蒙德矩阵的逆矩阵的方法较为复杂，具体可参考这篇文章，但是在 FFT 中，我们只需要计算当 $x = \omega_n$ 时的逆矩阵，求解过程依旧繁琐，下面直接给出结果：

$$\mathbb{F}^{-1} = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega_n^n & (\omega_n^n)^2 & ... & (\omega_n^n)^n \\ 1 & \omega_n^{n-1} & (\omega_n^{n-1})^2 & ... & (\omega_n^{n-1})^n \\ ... & ... & ... & ... & ... \\ 1 & \omega_n^1 & (\omega_n^1)^2 & ... & (\omega_n^1)^n \end{bmatrix}$$

ω_n^i 与 ω_n^{n-i} 互为倒数，所以只需改变函数的参数就可以把 FFT 变成 IFFT。

需要注意的是，不管是 FFT 还是 IFFT，为了方便递归操作，n 的值都应该是 2 的整数幂，如果不是需要，需要向高位补充 $a_n = 0n$，直到 n 等于 2 的整数幂为止。

伪代码：

Algorithm 1 快速傅里叶变换和逆变换

Input: n，复数数组 $a*$ 表示 $f(x)$，$type$ 是否为逆变换

```
function FFT(n, a*, type)
    if n = 1 then
        return
    end if
    for i = 0; i <= n;i+ = 2 do
        a1[i/2] ← a[i]
        a2[i/2] ← a[i + 1]
    end for
    // 把数组按奇偶分开
    FFT(n/2, a1, type)
    FFT(n/2, a2, type)
    Wn ← complex(cos(2π/n), type ∗ sin(2π/n))
    // type = 1 表示求 FFT，type = −1 表示求 IFFT
    w =← complex(1, 0)
    for i = 0 → n/2 do
        a[i] ← a1[i] + w ∗ a2[i]
        a[i + n/2] ← a1[i] − w ∗ a2[i]
        w ← Wn ∗ w
    end for
end function
```

5. 蝴蝶变换优化

由于普通的 FFT 动态需要不断复制数组和排序，动态申请和删除了 $O(nlog(n))$ 的空间，造成了大量的时间浪费，考虑是否能将递归求解转化为迭代求解。可以先模仿递归，把这些系数在原数组中「拆分」，然后再倍增地去「合并」这些算出来的值。

对于「拆分」可以使用位逆序置换实现。对于「合并」，使用蝶形运算优化可以做到只用 $O(1)$ 的额外空间来完成。

「拆分」:

以 8 项多项式为例，模拟从上至下拆分的过程:

- 初始序列为 $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7$
- 一次二分后 $x_0, x_2, x_4, x_6, x_1, x_3, x_5, x_7$
- 两次二分后 $x_0, x_4, x_2, x_6, x_1, x_5, x_3, x_7$
- 三次二分后 $x_0, x_4, x_2, x_6, x_1, x_5, x_3, x_7$
- 将下标用二进制表示 $000, 100, 010, 110, 001, 101, 011, 111$
- 将二进制数左右翻转 $000, 001, 010, 011, 100, 101, 110, 111$

可以发现最终序列的下标就是初始序列 a 下标的二进制翻转，我们称这个变换为位逆序置换。因此我们很容易就能求出最终数组的排序，由此递推回去得到最终答案。

只需将数列按照下标二进制翻转后的大小排序，得到序列 b，递归倒数第二层对于 a 来说就变成了 b 相邻两个数进行合并，往上以此类推。

现在的问题就变成了如何快速的求出一个二进制数的翻转形式，对于 x 的二进制翻转 $R(x)$，可以在 $O(n)$ 的时间内从小到大递推实现，设 $len = 2^k$，其中 k 表示二进制数的长度，现要求序列 $R(0), R(1), ..., R(n-1)$。考虑从小到大递推，将 x 右移一位得到的 $\lfloor \frac{x}{2} \rfloor$，其二进制翻转值是已知的，因此只需要把 x 右移一位，再进行翻转，再右移一位，并对翻转后最高位的结果进行计算，即可得到 $R(x)$, 即

$$R(x) = \lfloor \frac{R(\lfloor \frac{x}{2} \rfloor)}{2} \rfloor + (x \ mod \ 2) \times len$$

举例：$k = 5, len = 2^5 = (10000)_2$，翻转 $(11001)_2$，由于已经知道了 11001 右移一位后 01100 的翻转，$R(01100) = 00110$，右移一位得到了不计算最高位后的翻转值 00011，然后根据翻转前最低位的 0/1 判断最高位是 0/1，得到 10011。

至此，我们可以很轻松的写出代码。

Algorithm 2 求序列翻转序列 $R(x)$

```
for i = 0 → n − 1 do
    R[i] ← R[i >> 1] >> 1
    if i&1 then
        R[i] ← R[i]|n >> 1
        如果最后一位是 1，则翻转成 n/2
    end if
end for
for i = 0 → n − 1 do
    if R[i] > 1 then
        swap(a[i], a[R[i]])1
        交换原序列下标和新序列下标
    end if
end for
```

「合并」:

使用位逆序置换后，对于给定的 n, k: $G(\omega_{n/2}^k)$ 的值存储在数组下标为 k 的位置，$H(\omega_{n/2}^k)$ 的值存储在数组下标为 $k + \frac{n}{2}$ 的位置。

$f(\omega_n^k)$ 的值将存储在数组下标为 k 的位置，$f(\omega_n^{k+n/2})$ 的值将存储在数组下标为 $k + \frac{n}{2}$ 的位置。

因此可以直接对对应下标进行覆写操作，不需再额外开数组保存值。

所以蝶形运算可改为:

- 令段长度为 $s = \frac{n}{2}$
- 同时枚举序列 $\{G(\omega_{n/2}^k)\}$ 的左端点 $l_g = 0, 2s, 4s, \cdots, N - 2s$ 和序列 $\{H(\omega_{n/2}^k)\}$ 的左端点 $l_h = s, 3s, 5s, \cdots, N - s$;
- 合并两个段时，枚举 $k = 0, 1, 2, \cdots, s - 1$，此时 $G(\omega_{n/2}^k)$ 存储在数组下标为 $l_g + k$ 的位置，$H(\omega_{n/2}^k)$ 存储在数组下标为 $l_h + k$ 的位置;
- 使用蝶形运算求出 $f(\omega_n^k)$ 和 $f(\omega_n^{k+n/2})$，然后直接在原位置覆写。

伪代码：

Algorithm 3 快速傅里叶变换和逆变换

Input: $a*$ 为经过翻转后的序列

function FFT$(n, a*, type)$
  for $mid = 0;\ mid < n; mid <<= 1$ **do**
    $Wn \leftarrow complex(cos(2\pi/n), type * sin(2\pi/n))$
    $w =\leftarrow complex(1, 0)$
    $r \leftarrow mid << 1$
    $//r$ 代表区间右端点，j 枚举当前位置
    for $j = 0;\ j < n; j+ = r$ **do**
      for $k = 0 \rightarrow mid$ **do**
        $x \leftarrow a[j + k]$
        $y \leftarrow a[j + mid + k]$
        $a[j + k] \leftarrow x + y$
        $a[j + mid + k] \leftarrow x - y$
        $w \leftarrow Wn * w$
      end for
    end for
  end for
end function

1.2.3 快速数论变换 NTT

数论变换（number-theoretic transform, NTT）是离散傅里叶变换（DFT）在数论基础上的实现，快速数论变换（fast number-theoretic transform, FNTT）是快速傅里叶变换（FFT）在数论基础上的实现。

由于快速傅立叶变换是基于复数进行运算的，运算过程中涉及大量的浮点数运算，计算量和误差较大。在前置知识中我们知道了原根和单位根在性质上相似，因此在知道模数的前提下，可将复数 ω_n 转化为原根 g_N 进行变换，这就是数论变换。

在 DFT 与 FFT 的基础上，将复数加法与复数乘法替换为模 p 意义下的加法和乘法，一般大小限制在 0 到 $p-1$ 之间；将本原单位根改为模 p 意义下的相同阶数的本原单位根，阶数为 2 的幂，即可得到 NTT 与 FNTT。

在代码上，只需要将 ω_n 修改为 g_n 即可，其中 $g_N = g^{\frac{p-1}{N}}$，当求 NTT 时，g 为模 p 的原根，当求 INTT 时，g 为模 p 的原根对 n 的逆元。

2 密码学前置知识

全同态加密（Fully Homomorphic Encryption，FHE）依赖于**数论变换（Number Theoretic Transform，NTT）**以实现更加迅速的密文运算，但并不是所有的全同态加密过程都需要 NTT 的参与。如基于格密码学的，限于整数多项式环的 FHE 密文计算，需要 NTT 来优化多项式的乘法运算；但如普通的限于实数的 FHE 密文计算，就不需要 NTT 加速运算了。需要 NTT 加速的后量子密码 Kyber 算法和全同态加密算法 BFV，可以参阅目录，跳转查看详情。

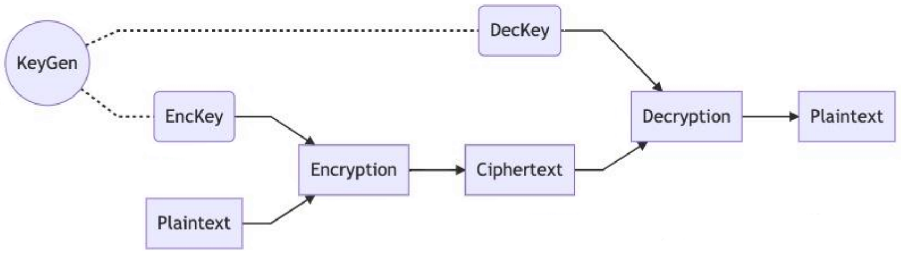

图 2.1: 加密体系框架

2.1 同态加密基础概念及发展历史

2.1.1 密码学的基础概念

1. 加密体系框架 在温故 FHE 的历史之前，我们先回顾一下最基础的**加密体系框架**。如图2.1所示。如果要构建一个加密系统，往往都需要一个密钥（Key）。通过这个密钥，我们可以一头把明文的信息加密成密文，然后在另一头通过密钥再把密文变回原来的样子。如果没有这个 Key 的话，其他的人很难知道我们到底传递了什么信息。[3]

所有的加密体系，如果用比较正式的描述方法，无疑是做了三件事：

1. 通过一个**生成密钥**算法 $KeyGen$, 来随机生成一对用于加密和解密的密钥 $(EncKey, DecKey)$。
2. 加密方通过加密密钥 $EncKey$ 和加密算法 $Encryption$ 来**加密原文** $Plaintext$，最后得到密文 $Ciphertext$。
3. 解密方可以通过解密密钥 $Deckey$ 和解密算法 $Decryption$ 来**解密密文**，最后还原回来原来的原文。

2. 对称与非对称加密体系 密码学两大常见的加密体系为**对称加密体系**和**非对称加密体系**。我们这里描述的这三个步骤，其实通用于任何加密体系。如果是对称的，那么加密和解密用的密钥就是一样的。如果是非对称的体系的话，那么加密用的就是公钥（Public Key），然后解密用的是不一样的私钥（Private Key）。

3. 同态的概念 但是在众多的加密算法当中，有一类算法生成的密文有一种特殊的同态属性：假如我们使用加密算法 Enc 得到了数字 1 的密文 $ct1$，然后我们又得到了数字 2 的密文 $ct2$ 。这个时候，如果我们把密文相加起来，有如下优良的性质，$ct1 + ct2 = Enc(1) + Enc(2) = Enc(1 + 2)$ 对于这种属性，我们称之为**加法同态**。意思就是说，**加密过后的密文与以前的原文保持着一种微妙的代数关系。如果我们把密文累加起来，我们就可以获得把原文相加起来加密过后的新密文**。同理可得，加法同态至于，还存在着乘法同态的加密算法。一个**乘法同态**的算法生成的密文，我们可以相乘起来，然后获得原文之间相乘之后的结果所对应的密文。整个过程中，我们不需要知道任何和密钥有关的信息，纯粹只是把看似像随机乱码的密文组合起来。

2.1.2 同态加密的分支

Topic. 同态加密体系的分类 如果系统性的来看，同态加密体系大致上被分为四类：部分同态、近似同态、有限级数全同态与完全同态。概括起来看，所有的算法可以归结为如图2.2。

1. 部分同态加密 (Partially Homomorphic Encryption) 同态加密发展的最初级的阶段，定义就是密文只有一种同态特性，只支持加法同态或乘法同态的其中一种。在**安全代理计算**的场景下看，

	类别		支持密文加密文的次数	支持密文乘密文的次数	举例
半同态	加法同态		∞	0	Paillier
半同态	乘法同态		0	∞	El Gamal
全同态	Somewhat 同态*	基于ECC	∞	1	BGN
全同态	Somewhat 同态*	基于 (R)LWE	∞	L≥1	BFV、BGV
全同态	全同态 with bootstrapping		∞	∞	BFV、BGV

图 2.2: 同态加密算法分支汇总

假如我们有私密输入 $f(x_1, x_2, ..., x_n)$，然后我们希望云端可以帮我们计算，那么我们可以把云端对于这些输入做的**计算使用函数** F 来表示。

假如说我们可以通过一个加法同态加密的算法来计算 F 的话，那么代表了这个函数 F 肯定就只能包含私密输入 x_i 的任意线性组合（加法运算）。一个可行的例子就是把各项私密输入乘以一个常数，然后相加起来：$f(x_1, x_2, ..., x_n) \to c_1x_1 + c_2x_2 + ... + c_nx_n$

但是如果我们想让两个输入 x_i, x_j 相乘起来的话，那么上述的加法同态算法就无能为力了。F 只能是所有私密输入的线性组合。

常见的加法同态加密算法就是基于循环群 $\mathbb{G}$ 的 ElGamal 加密算法。

ElGamal 是基于 Diffie-Hellman 密钥交换协议为基础而产生的一个非常方便的公钥加密算法，采用的是循环群的特性。由于篇幅的原因，我们就在这里不详细解释循环群的定义了，我们只需要知道每一个群都可以找到一个生成元（generator）$g \in \mathbb{G}$，然后这个生成元可以进行幂的计算。

ElGamal 加密实现方法大致如下：

1. 首先我们找到一个生成元 $g \in \mathbb{G}$，然后随机选择一个整数 x。我们把 g^x 作为公钥 pk，然后把 x 作为私钥 sk

2. 如果要加密一个消息 m 的话，那么我们就再随机选择一个整数 y，然后我们就可以输出密文。$ct \leftarrow (v \leftarrow g^y, e \leftarrow pk^y \cdot g^m)$。注意这个密文分为 v, e 两段。

3. 当我们需要解密一个密文 ct 的时候，我们只需要首先计算 $w \leftarrow v^x = g^{yx} = pk^y$，然后就可以轻松的还原出原文 $m \leftarrow log_g(e/w)$。。

看到这个加密体系的加密方式之后，由于都是幂运算，我们可以发现 ElGamal 潜在的加法同态的特性。

同理我们也可以应用于乘法同态加密的算法上——F 就只能把所有的私密输入相乘起来，但是没有办法做任何线性组合（加法）。乘法同态的例子其实也非常常见，**RSA 加密就是一个乘法同态的系统**。

RSA 加密的实现方法大致如下：

1. 首先找到一个很大的数字 $N = p \cdot q$，并且 p, q 为质数。然后找到一组数字 e, d 使得可以满足 $e \cdot d = 1 \mod (p-1)(q-1)$ 。我们把 (N, e) 作为公钥，(N, d) 作为私钥。

2. 如果要加密一个消息 m，我们只需要输出密文 $ct \leftarrow m^e \mod N$。

3. 如果需要解密一个消息 ct，我们只需要还原出原文 $m \leftarrow ct^d \bmod N$。

2. 近似同态加密（Somewhat Homomorphic Encryption） 就如同我们在上一段所说，如果我们又想让私密输入相乘，又想得到它们之间的线性组合的话，单纯的部分同态加密算法（RSA，ElGamal）是无法完成的。所以我们就需要来到下一阶段。

部分同态加密的下一阶段是近似同态加密，这一阶段距离我们想要实现的全同态更近了一步。如果我们有近似同态加密算法的话，那么我们就可以在密文上同时计算加法与乘法了。但是需要注意的是，正因为这一阶段是近似同态（Somewhat Homomorphic）的，所以可以做的加法和乘法次数非常有限，可以计算的函数 F 也在一个有限的范围内。

近似同态加密这一阶段常见的例子并不多，如果说最好理解的例子的话，那就应该是基于配对（Pairing）的循环群加密算法了。

上文我们简单的讨论过基于普通循环群的 ElGamal 加密算法。我们都知道这一算法是加法同态的，也就是说可以得到任意密文之间的线性组合。事实上，在某些特殊的循环群中，我们还可以找到一些薄弱的乘法同态性质。

配对（Pairing）是基于某些特有的椭圆曲线循环群可以进行的一种特殊运算，我们用 $e(\cdot,\cdot)$ 来表示。具体 Pairing 做的事情，就是把两个循环群中的值映射到第三个循环群中：

$$e(g^x \in \mathbb{G}, g^y \in \mathbb{G}) \rightarrow g_T^{xy} \in \mathbb{G}_T$$

这样一来，我们就变相的得到了前两个值的幂之间的乘积组合！再搭配 ElGamal 加密中可以把两个值的幂做线性组合的属性的话，那么我们就得到了一个全同态的系统了。

事实上，**现实并没有那么美好，因为 Pairing 这一特殊属性并不会出现在所有的循环群当中**。所以如果我们把两个可以做 Pairing 的群中的值通过 Pairing 相乘起来，映射到了一个新的群 $\mathbb{G}_T$ 当中之后，那么新的群并不一定能找到对应的 Pairing 属性！

这也就是说，通过拥有 Pairing 属性的循环群，我们只能做非常有限的乘法计算。假如说我们当前的群 $\mathbb{G}$ 支持 Pairing，但是新的映射群 $\mathbb{G}_T$ 并不支持任何 Pairing，那就代表了如果我们要基于当前的体系进行同态加密运算，可以运算的函数 F 虽然可以包涵任意的线性组合，但是只能包涵最多一层乘法在里面。

这一局限性证明了这个系统是近似同态的，因为我们不能计算任意逻辑和深度的函数 F。

3. 有限级数全同态加密（Fully Leveled Homomorphic Encryption） 来到下一个阶段之后，我们距离全同态的目标更进一步了。

这一阶段被称之为有限级数全同态加密。在这一阶段的话，我们已经可以对密文进行任意的加法乘法组合了，没有任何对于次数的局限性。

但是之所以被称之为有限级数全同态的原因是，这个阶段的算法会引入一个新的复杂度上限 L 的概念，这一复杂度上限约束了函数 F 的复杂度。如果我们可以把 F 用二进制电路 C 来表示的话，那么 C 的深度和大小一定要在 L 的范围之内，即：$|C| \leq L$

也就是说，如果 L 相对来说比较大的话，那么我们就可以进行各种各样较为简单（低复杂度）的同态运算了。有限级数同态加密的算法也是下一阶段全同态加密的奠基石，当我们实现了 L 复杂度以内的全同态之后，实现完全同态也不远了。

我们可以了解一下这个复杂度的上限是怎么来的。首先，我们可以想象一下，假如有一个全同态加密的算法，可以对密文进行任何的加法与乘法的运算。但是这个算法在加密的时候会在密文里面加入一定的随机噪音。

当噪音早可控范围内的时候，那么解密算法就可以很容易从密文还原回原文。但是当我们叠加密

文在一起进行同态计算的时候，每一个密文里面自带的噪音会被叠加扩大。如果我们只是对密文进行比较简单的逻辑，那么叠加起来的噪音还在一个可以接受的范围。但是如果我们过于复杂地堆叠密文在一起，那么一旦噪音的范围超过了临界值，那么就会彻底的覆盖掉原来的原文，从而导致解密会失败。

在这个场景中，这个同态加密系统可以接受的噪音上限转换为叠加的逻辑的复杂度的话就是了。这就是为什么我们只可以进行复杂度小于 L 的计算，因为一旦复杂度超过 L，我们就再也无法还原回原来的原文了。

4. 全同态加密（Fully Homomorphic Encryption，FHE） 千呼万唤使出来，最后就到我们拭目以待的全同态加密（FHE）了。就像名字所说的一样，一个全同态加密的系统没有任何计算方法的限制，我们可以在没有密钥的情况下，把密文任意的组合起来，形成新的密文，并且新的密文，无论计算的复杂度，都可以完美的被还原成原文。

当我们达到这一阶段的时候，之前提到的安全代理计算就变得可行了。如果可以找到一个效率比较高的全同态加密体系的话，我们可以安全的把所有本地的计算全部代理到云端，并且不会泄露任何一丁点数据！

全同态加密体系的定义在开始下文对于全同态历史的讨论之前，我们可以系统性的定义一下全同态系统的正式定义。一个全同态加密系统，一共拥有四个算法：

1. **密钥生成算法**：$KeyGen(1^\lambda) \to sk$ 生成加密与解密需要用到的密钥 sk。为了简单表示，我们这里假设这个加密系统是对称的（即加密密钥等于解密密钥）。
2. **加密算法**：$Enc(sk, m \in 0, 1^{|M|}) \to ct$ 把原文 m 加密成密文 ct。
3. **解密算法**：$Dec(sk, ct) \to m$ 把密文 ct 还原为原文 m。。
4. **运算算法**：$Eval(F, ct_1, \ldots, ct_l) \to \hat{ct}$ 把 l 个密文组合起来，通过一个二进制逻辑电路 F，最后得到组合的密文 $\hat{ct}$，使得 $Dec(sk, \hat{ct}) = F(m_1, \ldots, m_l)$ 。

我们可以看到，最后一步的运算算法 $Eval$ ，就是全同态加密的精髓了。回到上文的例子，如果我们想让云端的服务器替我们进行一些机器学习的模型分析，那么在这里，我们只需要把模型分析预测的过程转换为二进制逻辑电路 F，然后我们提供加密过后的数据样本 $ct_1, \ldots, ct_l$ ，然后云端就可以通过 $Eval(F, ct_1, \ldots, ct_l)$, 得到加密的最后预测的结果。

2.1.3 全同态加密体系的性质

现在我们来看看这个系统的属性（Properties）。

1. 这个体系必须得是正确的（Correctness）. 也就是说，如果我们任意选择一个电路 F，并且任意选择一组原文消息 $m_1, \ldots, m_l$。如果我们拥有一开始 $KeyGen$ 算法生成的密钥的话，那么：

$$Dec(sk, Eval(F, Enc(sk, m_1), \ldots, Enc(sk, m_l))) = F(m_1, \ldots, m_l)$$

2. 这个系统需要达到语义安全（Semantic Security）。语义安全我们可以理解为，如果我们拥有任意两个不同的原文所对应的密文，那么我们是无法区分到底哪个密文是对应了哪个原文的：

$$\forall m_0, m_1 : \{Enc(k_{enc}, m_0)\} \approx_c \{Enc(k_{dec}, m_1)\}$$

语义安全的主要意义在于旁观者无法区分两条加密的消息。那么如果有入侵者窃听网络，看到了我发

出的加密信息，只要我使用的加密体系是语义安全的，那么我就可以确信入侵者无法从密文中得到关于加密内容的任何信息。

3. 为了让全同态加密体系变得有实际的使用意义，我们必须还得加一条额外的规定：简短性（Compactness）。简短性是什么意思呢？其实很简单，就是说 $Eval$ 这个算法输出的结果必须要在一个限定范围之内。如果要加一个硬性条件的话，就是 $Eval$ 这个算法的输出结果大小必须要独立于二进制电路 F 的大小。也就是说，就算 F 再大，$Eval$ 的结果大小也不会有所改变：

$$\forall F, sk, ct_i \leftarrow Enc(sk, m_i)$$
$$\mid Eval(F, ct_1, \ldots, ct_l) \mid = poly(\lambda)$$

为什么需要加上这个简短性的特性呢？因为如果没有这个要求，我们可以非常轻易的做出一个毫无意义的（作弊的）全同态系统：

1. 密钥生成、加密算法可以任意选择一个语义安全的对称加密算法
2. $Eval(F, ct_i) \rightarrow (F, ct_i)$: 运算算法 $Eval$ 要做的事情很简单，直接把对于 F 的描述和原来的密文 ct_i 全部输出到新的密文 $\hat{ct}$ 当中。
3. $Dec(sk, (F, ct_i)) \rightarrow F(Dec(sk, ct_1), \ldots, Dec(sk, ct_l))$: 最后在解密的时候，先把密文里所有的 ct_i 全部依次解密回原文，然后再根据对 F 的描述手动跑一下 F 得到原来的结果。

如果没有对与 $Eval$ 输出大小的限制的话，那么我们反复叠加多次 $Eval$ 之后，得到的密文大小将会越来越大。最后解密的时候，只需要把所有的原始密文解开，然后算一下 F 就行了。这就好比是一个用户把自己的健康信息加密了，让医院去判断他有没有生病，医院直接原封不动的把密文送回来，然后把自己的整套数据模型加上医疗课本也发回来，跟用户说，你自己去算吧，是一个意思。

这一类全同态系统还有一个更大的弊端，也就是最后得到的密文并无法完全掩盖住运算电路 F 的具体细节。在这个医院的使用场景中，有可能医院自己最值钱的就是这套数据分析系统。如果白白的把自己的 F 发回给了用户，那自己的辛苦劳动成果就被别人轻易窃取了。

综上所述，只要满足正确、语义安全、简短这三个要素，我们就拥有一个有意义（Non-trivial）的全同态加密体系了。

2.1.4 同态加密算法历史

在开始学习全同态加密算法到底是怎么实现的之前，我们不妨来看看全同态加密这个概念到底是怎么来的。其大致的历史脉络如图2.3所示.

FHE（全同态）的概念其实在上世纪 70 年代末就已经被提出了。1978 年，密码学界的几个大牛 Rivest，Adleman 和 Dertouzos 在他们的论文 On Data Banks and Privacy Homomorphisms 中第一次提到了对于密文进行一定的计算，可以间接地对原文进行操作的系统构想。到后来这一想法就被重新总结命名为全同态加密了。

由此可见，全同态加密这一概念已经被提出了很久了。令人惊讶的是，1976 年，也就是论文发表的两年前，Diffle-Hellman 密钥交换协议才刚刚被提出！由此可见密码届大牛的想象力还是非常丰富的。

当 FHE 的概念被提出来之后，整个学术界都为之所动，开始了长达几十年的搜索，试图找到一个拥有全同态性质的完美算法。但是这几十年下来，人们试遍了所有可以想到的选择，但是找不到一个又能满足全同态所有条件，并且安全性可以被轻易证实的选项。

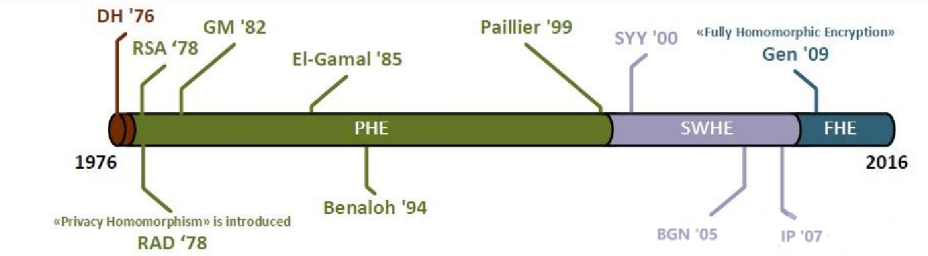

图 2.3: 同态加密算法历史

直到 2009 年，在斯坦福读书的 PhD Craig **Gentry** 突然灵光一现，攻破了 FHE 算法的难关。在他的博士毕业论文中，他第一次给出了一个合理并且安全的全同态加密系统！这一系统基于理想格（ideal lattice）的假设。**Gentry09 提出来的全同态系统，我们往往称之为第一代全同态加密系统**。

在 Gentry 的论文中，他还提到了一个至关重要的概念叫做 Bootstrapping。Bootstrapping 是一种对于密文的特殊处理技巧，处理过后竟然可以把一个噪音接近临界值的密文“重新刷新”成一个噪音很低的新密文。通过 Bootstrapping，一个有限级数的系统的噪音可以永远不超过临界值，从而变成了全同态的系统。这样一来，我们就可以同态计算任意大小的了。

在 Gentry 的重大突破之后，整个密码圈又一次陷入了疯狂，大家都开始争相基于 Gentry 提出的想法寻找更加高效率和全能的全同态体系。

在 2011 年的时候，两位大佬 Brakerski 和 Vaikuntanathan 提出了一个新的全同态加密体系，这一体系基于格（lattice）加密的另一种假设 **Learning With Errors (LWE)**。在同一年，Brakerski，Gentry 与 Vaikuntanathan 这三人一起把这个体系做完了，并且正式发表出来。他们发明的全同态系统简称为 BGV 系统。BGV 系统是一个有限级数的同态加密系统，但是可以通过 Bootstrapping 的方式来变成全同态系统。BGV 系统相比起 Gentry09 提出的系统，使用了更加实际一点的 LWE 假设。**一般来说我们都把 BGV 系统称之为第二代全同态加密系统**。

2013 年，Gentry 又卷土重来了。Gentry，Sahai 和 Waters 三个大佬推出了新的 GSW 全同态加密系统。GSW 系统和 BGV 相似，本身具有有限级数全同态性质，基于更加简单的 LWE 假设，并且通过 Bootstrapping 可以达到全同态。**我们一般把 GSW 系统称为第三代全同态加密系统**。

2013 年之后，密码圈基本上就百花齐放了。基于原来的三代全同态系统之上，出现了各种各样新的设计，致力于优化和加速 BGV 与 GSW 系统的运行效率。IBM 基于 BGV 系统开发了一个开源的全同态运算库 HElib，并且成功的移植到各大移动平台上。与此同时，还有另外一个开源项目 TFHE 也非常值得注意。TFHE 是基于 GSW 系统，又加以了各种优化与加速，现在也非常的有名。

在开发传统的全同态库之外，也有很多团队在研究如何通过 GPU，FPGA，ASIC 等**异构硬件来更好的加速全同态加密算法的计算**。比如 cuFHE 就是一个比较有名的基于 CUDA 的 GPU 加速全同态加密系统。

站在今天的角度上，一路看来，全同态体系的大门被 Gentry 大神敲开已经过去了 11 年了。现在业界对于 FHE 的研究百花齐放，不少人都在不同的角度和应用需求上在研究全同态系统。直到今天，我们已经拥有了多种可行的 FHE 实现方法，但是现在大家还在不断追求的是 FHE 系统运转的效率。拿现在最前沿的 FHE 库来说，在移动平台上同态计算一些比较简单的逻辑可能要少则花上几十毫秒，多则花费几十秒的时间。这些时间单位对于计算机系统来说是极其缓慢的。

如何可以让 FHE 系统更加高效率的在异构平台上运行，仍然是一个未解之谜。如果这道难题一旦被解决了，那么把所有的电脑运算都转为全同态，代理在第三方的云端上进行计算，都是伸手可得的未来。

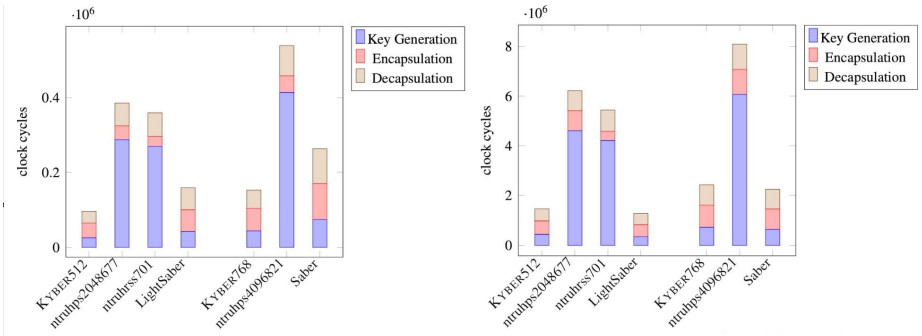

Figure 1. KEM Benchmarks on x86-64 processors with AVX2 extensions

Figure 3. KEM Benchmarks on ARM Cortex-M4 processor

图 2.4: Kyper 算法分别在 x86 和 Arm 架构上的性能优势 (计算)

2.2 格密码学（Lattice Cryptography）

2.2.1 传统密码学困境

随着量子计算的发展，传统的加密技术面临着前所未有的挑战。**上文提到的传统的加密算法，如 RSA 和椭圆曲线加密，依赖于大数分解和离散对数等难题的计算复杂度，从而保证了信息的安全性。然而，随着量子计算的崛起，这些复杂度被量子计算机可能迅速解决的能力所打破，这就催生了对抗量子计算的密码学——格密码学的兴起。**

格密码学基于格的数学结构，利用格的难题来设计新的加密方案，以应对量子计算的挑战。在格密码学中，加密方案的安全性建立在格问题的复杂度上，这些问题通常包括基于格的难题，如**最短向量问题（SVP）**和**最密背包问题（CVP）**，这些问题在经典计算机上也很难解决。

与传统的基于数论问题的加密算法相比，格密码学具有更强的抵抗量子计算攻击的能力。因为量子计算机在解决基于格的问题上并没有明显的优势，而且已有的量子算法在这些问题上也没有实质性的突破，**具有抗量子的密码体系被称为后量子计算密码学。**

2.2.2 后量子计算密码学（Post-Quantum Cryptography）

2022.7.5，在经历了数个月的“拖延“后，NIST **终于公布了通过第三轮 NIST 后量子密码算法标准征集的 4 个算法。这** 4 个算法将开启正式的标准化进程，并预期在 2024 年完成正式的标准。[1]

征集入选的加密算法 CRYSTALS-KYBER 就是基于格密码（Structured-lattices）的。

Kyber 是基于 Module Learning with Errors (MLWE) 问题的 KEM 算法，是基于 Structured lattices 的。Kyber 提供了 IND-CPA 安全和 IND-CCA 安全版本的加密算法，其中 CCA 版本是在 CPA 版本基础上加入了常用的 Fujisaki-Okomoto (FO) 变换得到的。

Kyber 的基础构造是基于环 $R = \mathbb{Z}[x]/(X^{256}+1)$ 上的，并且 module rank $k = 2, 3, 4$ 以提供不同等级的安全性（分别对应于 NIST 的 Security category 1, 3, 5，大致对应于 128/192/256-bit security。Kyber 的 modulus 选择为 3329，这使得算法的通信开销比其他算法更有竞争力，并且更容易实现更快速的多项式乘法。

除安全性外，NIST 认为 Kyber 在多种平台的软硬件实现上都能达到很好的性能，可以很好的嵌入到大多数现有的互联网协议/密码算法应用中，因此选择了 Kyber 作为后量子加密算法的标准。Kyber 与其他被淘汰的算法对比计算性能，如图2.4所示;Kyber 与其他被淘汰的算法对比计算 + 通讯性能，如图2.5所示.

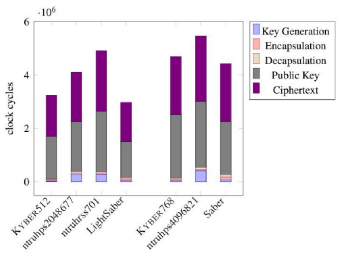

Figure 2. KEM Benchmarks on x86-64 processors with AVX2 extensions with 2000 cycles/byte transmission costs

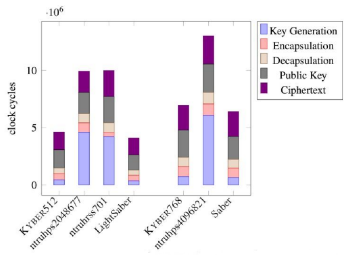

Figure 4. KEM Benchmarks on ARM Cortex-M4 processor with 2000 cycles/byte transmission costs

图 2.5: Kyper 算法分别在 x86 和 Arm 架构上的性能优势 (计算 + 通讯)

Candidate	Claimed Security	Public key	Private key	Ciphertext
Classic McEliece348864	Level 1	261 120	6 492	128
Classic McEliece460896	Level 3	524 160	13 608	188
Classic McEliece6688128	Level 5	104 992	13 932	240
Classic McEliece6960119	Level 5	1 047 319	13 948	226
Classic McEliece8192128	Level 5	1 357 824	14 120	240
KYBER512	Level 1	800	1 632	768
KYBER768	Level 3	1 184	2 400	1 088
KYBER1024	Level 5	1 568	3 168	1 568
NTRU-HPS2048677	Level 1	930	1 234	930
NTRU-HRSS701	Level 1	1 138	1 450	1 138
NTRU-HPS4096821	Level 3	1 230	1 590	1 230
NTRU-HPS40961229	Level 5	1 842	2 366	1 842
NTRU-HRSS1373	Level 5	2 401	2 983	2 401
Light Saber	Level 1	672	832	736
Saber	Level 3	992	1 248	1 088
Fire Saber	Level 5	1 312	1 654	1 472

图 2.6: 部分候选算法和 Kyber 的公私钥大小（bytes）、密文大小（bytes）比较

除此之外，Kyper 算法生成的密钥、密文长度精简，节省了大量的存储空间，各算法密钥长度对比可以参阅图2.6。

2.2.3 格密码在同态加密中的运用（BFV）

为了提升算法的复杂度，现代的许多同态加密算法已经基于格密码学开展，使得同态加密也具备抗量子的特性。这里以**同态加密算法 BFV（Brakerski/Fan-Vercauteren）**为例，BFV 是一种支持在加密数据上直接进行算术运算（如加法和乘法）的先进加密技术, 结合了格密码学中 LWE 问题，并且将问题难度从整数域扩展到了多项式环域。

1. 符号说明 我们用 Z_q 表示从 $(-q/2, q/2]$（q 为大于 1 的整数）的整数集。并且如果没有说明，所有的整数的操作（加法等等）都需要 $modq$，

$$\mathrm{v} \in \mathbb{Z}_{\mathrm{q}}^{\mathrm{n}} = \text{一个 n 维向量, 向量内的元素属于} \mathbb{Z}_{\mathrm{q}}$$

$$[.]_{\mathfrak{m}} = \bmod \mathfrak{m}\text{的操作}$$

$$< \mathrm{a}, \mathrm{b} >= \sum_{\mathrm{i}}^{\mathrm{n}} \mathrm{a_i} \cdot \mathrm{b_i} (\mathrm{mod\ q})$$

2. 多项式运算 (NTT 优化) 以 $\mathrm{Z_{11}[x]/ < x^4 + 1 >}$ 为例。

注意: 这个式子子的是一个多项式，其系数需要模 11，其多项式需要模 $x^4 + 1$.

多项式加法

$a(x) = 7x^3 + 4x^2 + 9$

$b(x) = x^3 + 10x^2 + 3x + 5$

$a(x) + b(x) = (7 + 1mod11)x^3 + (4 + 10mod11)x^2 + (3mod11)x + (9 + 5mod11)$

$a(x)+b(x)(modx^4 + 1) = 8x^3 + 3x^2 + 3x + 3$

多项式乘法

$a(x) = 5x^2 + 3$

$b(x) = 3x^3 + 4x^2$

$a(x) * b(x) = (15mod11x^5 + 20mod11x^4 + 9n$

$a(x) + b(x)(modx^4 + 1) = 9x^3 + x^2 + 7x + 2$

具体过程:$4x^5 + 9x^4 + 9x^3 + x^2 = (4x + 9) \cdot (x^4 + 1) + 9x^3 + x^2 + 7x + 2$

3 Learning With Error & Ring Learning With Error Learning With Error (LWE) was introduced by Regev in 2009 and can bedefined as follows: For integers $n > 1$ and $q > 2$, let's consider the following equations:

$$\langle s, a_1 \rangle + e_1 = b_1 (mod\ q)$$
$$\langle s, a_2 \rangle + e_2 = b_2 (mod\ q)$$
$$\cdots$$
$$\langle s, a_m \rangle + e_m = b_m (mod\ q)$$

where s and a_i are chosen independently and uniformly from $\mathbb{Z}_q^n$, and e_i are chosen independently according to a probability distribution over $\mathbb{Z}_q$and $b_i \in \mathbb{Z}_q$. **The LWE problem state that it's hard to recover s from the pairs** (a_i, b_i).

Ring-LWE 问题是 LWE 问题的变体，它也基于根据 (a_i, b_i) 恢复 s 的困难性，等式也基本相同，但是变量的取值空间从 $\mathbb{Z}_q^n$ 变成了多项式环

$$\mathrm{R_q = Z_q[x]/ < x^n + 1 >}$$

4. 构建一个全同态体系 我们将整个方案分解为基本功能、**密钥生成** $KeyGen$、**加密** $Encryption$、**解密** $Decryption$ **和评估** $Evaluate$。

密钥生成函数，其中 sk 是私钥，pk 是公钥，sk 是模 1 多项式环的二项分布，e 是模 1 多项式环的正态分布随机，a 是模 q 多项式环的均匀分布随机.

$$KeyGen(pk, sk) = \begin{cases} sk \in \mathbb{R}_1^n, \\ e \in \mathbb{R}_1^n, \\ a \in \mathbb{R}_q^n, \\ pk = ([-(a \cdot sk + e)]_q, a) \end{cases} \tag{7}$$

加密函数，其中 ct 为密文, m 为明文。其中 u 是模 1 多项式环的二项分布,e_1, e_2 是模 1 多项式环

的正态分布随机。

$$Encryption(pk, sk, ct) = \begin{cases} pk = ([-(a \cdot sk + e)]_q, a) \in \mathbb{R}_q \times \mathbb{R}_q \\ ct = (ct_0, ct_1) \in \mathbb{R}_q \times \mathbb{R}_q \\ \delta = \lfloor q/t \rfloor \\ ct_0 = [pk_0 \cdot u + e_1 + \delta \cdot m]_q \\ ct_1 = [pk_1 \cdot u + e_2]_q \end{cases} \tag{8}$$

解密函数 通过上方的构造我们可以发现，

$$pk_0 \approx -(pk_1 \cdot sk)$$

这也是我们解密的关键之处。那么我们可以构造如下式子就可以进行解密。

$$\begin{aligned} [ct_0 + ct_1 \cdot sk]_q &= [pk_0 \cdot u + e_1 + \delta \cdot m + (pk_1 \cdot u + e_2) \cdot sk]_q \\ &= [-(a \cdot sk + e) \cdot u + e_1 + \delta \cdot m + a \cdot sk \cdot u + e_2 \cdot sk]_q \\ &= [-a \cdot sk \cdot u - e \cdot u + e_1 + \delta \cdot m + a \cdot sk \cdot u + e_2 \cdot sk]_q \\ &= [\delta \cdot m - e \cdot u + e_1 + e_2 \cdot sk]_q \\ &= [\delta \cdot m + errors]_q \end{aligned}$$

再进一步化简，提取出明文 m, 再将结果返回到 $\mathbb{R}_t$ 空间。

$$\frac{1}{\delta} \cdot [ct_0 + ct_1 \cdot sk]_q = [m + \frac{1}{\delta} \cdot errors]_q \tag{9}$$

$$[\lfloor \frac{1}{\delta} \cdot [ct_0 + ct_1 \cdot sk]_q \rceil]_t = [\lfloor [m + \frac{1}{\delta} \cdot errors]_q \rceil]_t$$

注意这边有一个防止扰动过大的约束条件。

$$\text{condition: } errors \leq \frac{1}{2} \Leftrightarrow errors \leq \frac{q}{2t}$$

评估函数 接下来讨论如何计算加密的数据，当我们手上有密文时，我们可以对该密文进行加法或乘法，运算的对象可以是其他的密文或明文，我们试着添加明文的操作，这意味着我们将向我们的方案中添加将密文与整数 (明文) 相加或相乘的功能。为了简化问题，这边只说明明文+密文的同态。

加法同态 在加密的过程中，密文 ct 与原明文 $m1$ 之间的关系：

$$ct = ([pk_0 \cdot u + e_1 + \delta \cdot m_1]_q, [pk_1 \cdot u + e_2]_q)$$

当我们想要让密文加上另一个明文 m_2 时，$ct0'$：

$$ct_0' = [pk_0 \cdot u + e_1 + \delta \cdot (m_1 + m_2)]_q$$

我们只需要将新明文 m_2 放缩 δ 倍即可

$$add\_plain(ct, m2) = ([ct_0 + \delta \cdot m_2]_q, ct_1)$$

解密过程中与之前相比也没有添加新的误差:

$$[\lfloor \frac{1}{\delta} \cdot [ct_0 + ct_1 \cdot sk]_q \rceil]_t = [\lfloor [m_1 + m_2 + \frac{1}{\delta} \cdot (-e \cdot u + e_1 + e_2 \cdot sk)]_q \rceil]_t$$

因为误差不会增加也不需要考虑 BootStrapping 的问题，我们可以执行无限制次的加法。

乘法同态 ct 是密文，m_1 为原明文，我们想要乘上一个明文 m_2:

$$ct_0' = [pk_0 \cdot u + e_1 + \delta \cdot (m_1 \cdot m_2)]_q$$

当我们只对 ct_0 乘上 m_2 时，发现我们无法解码成功:

$$[ct_0' + ct_1 \cdot sk]_q = [pk_0 \cdot u \cdot m_2 + e_1 \cdot m_2 + \delta \cdot m_1 \cdot m_2 + pk_1 \cdot sk \cdot u + e_2 \cdot sk]_q$$

$$[ct_0' + ct_1 \cdot sk]_q = [-(a \cdot sk + e) \cdot u \cdot m_2 + e_1 \cdot m_2 + \delta \cdot m_1 \cdot m_2 + a \cdot sk \cdot u + e_2 \cdot sk]_q$$

$$[ct_0' + ct_1 \cdot sk]_q = [-a \cdot sk \cdot u \cdot m_2 - e \cdot u \cdot m_2 + e_1 \cdot m_2 + \delta \cdot m_1 \cdot m_2 + a \cdot sk \cdot u + e_2 \cdot sk]_q$$

$-a \cdot sk \cdot u \cdot m2$ 和 $a \cdot sk \cdot u$ 无法抵消，正确的做法是对 ct_1 也乘上 m_2:

$$mul\_plain(ct, m2) = ([ct_0 \cdot m_2]_q, [ct_1 \cdot m_2]_q)$$

$$[ct_0 + ct_1 \cdot sk]_q = [-a \cdot sk \cdot u \cdot m_2 - e \cdot u \cdot m_2 + e_1 \cdot m_2 + \delta \cdot m_1 \cdot m_2 + a \cdot sk \cdot u \cdot m_2 + e_2 \cdot sk \cdot m_2]_q$$

$$[ct_0 + ct_1 \cdot sk]_q = [\delta \cdot m_1 \cdot m_2 - e \cdot u \cdot m_2 + e_1 \cdot m_2 + e_2 \cdot sk \cdot m_2]_q$$

解密，计算误差:

$$[\lfloor \frac{1}{\delta} \cdot [ct_0 + ct_1 \cdot sk]_q \rceil]_t = [\lfloor [m_1 \cdot m_2 + \frac{1}{\delta} \cdot (-e \cdot u \cdot m_2 + e_1 \cdot m_2 + e_2 \cdot sk \cdot m_2)]_q \rceil]_t$$

与加法对比，乘法明显的将误差增大了（误差项全乘上一个 m_2），意味着乘上一个比较大的密文时将无法解密成功。可以用 BootStrapping 的方法取解决这个扰动过大的问题。

3 数据集描述

数据集共包括三个部分，NTT&INTT 数据集（用来测试核心算法 NTT 的优化效果），同态加密数据集（用来测试优化对全同态加密的优化效果），安全向量内积数据集（用来测试全同态加密中安全向量内积的优化效果）。

3.0.1 NTT&INTT 数据集

优化 NTT 是优化全同态加密的最重要的组成部分，而 NTT 又是整个同态加密过程中最浪费时间的部分，所以我们优先测算并行优化 NTT 的时间。NTT 算法开发之初就是为了处理多项式乘法（卷积）的问题，具体涉及到 NTT 优化的地方参考“研究思路参考”中的“同态加密的 packing 机制”。

多项式乘法操作，涉及到两次 NTT 和一次 INTT 变换，但全同态加密时不仅有矩阵，还有单独的 NTT 和 INTT 操作，所以设计数据集时，针对单次 NTT 和 INTT 进行数据设计：PolyMulti.bin 是多项式的二进制文件，总共 2^25 个系数，每个系数大于等于 0 小于等于 9，代表 30000 项多项式，用来测试将系数表达式转换为点值表达式（单次 NTT）的时间。

3.0.2 同态加密数据集

同态加密的数据集分成两个文件，分别对应整数加密和浮点数加密。
HB1.din 对应整数加密，共 30000 个整数，大小在 1 10000。
HB2.din 对应浮点数加密，共 30000 个浮点，具体算法第 i 个数为 $\frac{1}{i}$，保留小数点后 6 位。

3.0.3 安全向量内积数据集

安全向量内积数据集在 InnerProduct.din 中，包含两个整数向量，每个向量的维数是 30000，每一维是一个 0 10000 的整数。

4 研究思路参考

4.0.1 NTT 的并行化优化

- GPU：用 GPU 加速 NTT 有三个办法。

  (1) 优化位逆序变换：位逆序变换是通过迭代的方式实现的，这种方式存在数据依赖，很难做到并行计算。可以使用 GPU 加速这一过程，为每个位序变换分配不同线程。

  (2) 共享原根幂：在每一轮递归操作中，每段区间的后半部分的原根幂集合与前半部分的原根幂值集合相同。例如，多项式项数长度为 n，第一层需要求出 $g^{i\frac{p-1}{n}}(i=0,1,...,\frac{n}{2})$，第二层需要求出 $g^{i\frac{p-1}{n/2}}(i=0,1,...,\frac{n}{4})$，以此类推。因此使用空间换时间的方式，每层开辟 $\frac{n}{2}$ 的空间存放可重复利用的原根幂。

  (3) 由于在计算每一层元素时，类似归并一样，计算上一层元素值时需要用到下一层的值，连续的两层之间存在数据依赖关系，造成了最外层枚举区间长度的循环无法使用并行优化，只能对内层的两个 for 循环并行。每次 GPU 并行计算的是一个层的所有 n 个元素，对于长度 n=2^m 来说，并行过程需要迭代 m 次。在其中的一次迭代过程中，只需要分配 $\frac{n}{2}$ 个线程，因为每个线程更新两个元素的值。

- SIMD：[4] 根据 FFT 算法的特点，在每一级的所有 FFT 蝶形单元中，前后两个待运算数据是等间隔的，并进行同样结构的基本蝶形运算。每个蝶形单元的数据间隔为 $\frac{n}{2}$，且两数之和存回第一个数据的原位置，两数之差与蝶形因子的积存回第二个数据的原位置，这个特性非常适合进行数据的并行处理，因此可以利用这一性质在向量处理器上基于 SIMD 实现 FFT 并行计算的方法。

- 多线程：基于多线程的并行算法的思想是：若计算机有 m 个处理器，[5] 则在程序中开创 m 个线程来计算，其中每个线程分担的负载都一样, 均为 $\frac{n}{m}$ 个，每个线程平均分担了计算任务。由于有 m 个处理器，操作系统每次都会分配 m 个时间片，每个线程都将获得时间片而不必阻塞以等待操作系统重新分配时间片，因而使用了多线程优化了并行效率。在 NTT 中需要使用一个矢量 d 分配将要计算的点的序号，矢量 d 中的值由主线程计算得出，并将其所对应的点分配到各个计算线程。这样一来, 就实现了主线程对分线程的计算负载的动态分配。

- MPI：传统串行方法使用存储在连续内存中的缓冲区对全体进行通信，因此必然需要在重新分配和转换步骤之间对数据进行重新对齐。可以利用子数组数据类型和 MPI-2 标准中的广义全对全分散/聚集 (generalized all-to-all scatter/gather) 来通信不连续的内存缓冲区，从而有效地消除了重新对齐数据的额外消耗时间。尽管不连续数据的通信通常较慢，但这一优化有效的提升了本地的时间优化。

4.1 同态加密编码优化

4.1.1 同态加密的 Packing 机制

第二代 FHE，包括 BGV，BFV，CKKS，目前在整体计算性能（不考虑 bootstrapping 自举操作）上是最优的。其中一个关键性技术就是 SIMD (Single Instruction Multiple Data)，也叫做 packing 机制。**通过 SIMD，可以将一个定义在 $\mathbb{F}_t$ 上的向量编码成 R_t 上的一个多项式。换句话说，SIMD 使得 FHE 可以直接操作一组向量**。依据中国剩余定理的同构性质，上多项式的加法和乘法等价于向量的对应元素加法（point-wise addition）和对应元素乘法（point-wise multiplication），因此，SIMD 编码使得并行化同态加法和同态乘法成为可能。FHE 的系统参数设定导致 SIMD 编码的向量维度较大，典型值为 65536。因此第二代 FHE 运算的并行度很高，比较适合真实场景中对同态计算速度要求高的情形。[2]

1.BGV, BFV 中的 SIMD 编码 (int) 首先讨论 SIMD 编码的基本形式，也就是 BGV 和 BFV 中使用的方法。设 ζ 是 $2N$ 阶原根，即不存在小于 $2N$ 的整数 x 使得 $\zeta^x = 1 \bmod t$，且 $\zeta^2 N = 1 \bmod t$。

x^N+1 可以因式分解为 $\prod_{i=0}^{N-1}(x-\zeta^{2i+1})$。依据中国剩余定理 (Chinese Remainder Theorem, CRT) 有:

$$R_t = \frac{\mathbb{Z}_t[x]}{x^N+1} = \frac{\mathbb{Z}_t[x]}{\prod_{i=0}^{N-1}(x-\zeta^{2i+1})} \simeq^{CRT} \prod_{i=0}^{N-1}\frac{\mathbb{Z}_t[x]}{x-\zeta^{2i+1}} = \prod_{i=0}^{i=N-1}\mathbb{Z}_t[\zeta^{2i+1}]$$

换句话说，R_t 上的任意一个多项式 $p(x) = \sum_i p_i x^i$ 都可以等价表示成它在 ζ^{2i+1} 上的估值，即 $p(\zeta^{2i+1})$ for $i \in [N]$ 。将该运算记作 $\sigma(p(x))$，用矩阵的形式表示如下:

$$\begin{bmatrix} m_0 \\ \vdots \\ m_{N-1} \end{bmatrix} = \begin{bmatrix} \zeta^{1\cdot 0} & \cdots & \zeta^{1\cdot(N-1)} \\ \vdots & \vdots & \vdots \\ \zeta^{(2N-1)\cdot 0} & \cdots & \zeta^{(2N-1)\cdot(N-1)} \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ \vdots \\ p_{N-1} \end{bmatrix}$$

这里用向量 $[m_0, \cdots, m_{N-1}]$ 表示多项式上的 N 个估值。这些估值在 SIMD 技术中就是需要进行编码的明文向量。也就是说，在 BGV/BFV 中，需要加密的信息是一个明文向量 $[m_0, \cdots, m_{N-1}]$ ，通过 SIMD 编码将其转换成 R_t 上的多项式 $p(x)$ ，进而对 $P(x)$ 进行正常的 RLWE 加密和进行相应的同态操作。容易看出 SIMD 编码实际上就是上面提到的多项式估值的逆运算，记作 σ^{-1} 。用矩阵的形式可以表示成:

$$\begin{bmatrix} p_0 \\ \vdots \\ p_{N-1} \end{bmatrix} = \begin{bmatrix} \zeta^{1\cdot 0} & \cdots & \zeta^{1\cdot(N-1)} \\ \vdots & \vdots & \vdots \\ \zeta^{(2N-1)\cdot 0} & \cdots & \zeta^{(2N-1)\cdot(N-1)} \end{bmatrix}^{-1} \cdot \begin{bmatrix} m_0 \\ \vdots \\ m_{N-1} \end{bmatrix}$$

在密码实践中，通常使用数论变换 (Number Theoretic Transform, NTT) 的一种变式快速实现 SIMD 编码，如上矩阵。

2.CKKS 中的 SIMD 编码 (float) 现在讨论 CKKS 的 SIMD 编码方法。和 BFV/BGV 的 SIMD 最大的区别在于，**CKKS 的输入不在是一个整数型的向量，而是一个浮点型（更准确地说，复数域）的向量 $\mathbf{z} \in \mathbb{C}^{N/2}$** ，通过 SIMD 最终将其编码为整数环上的多项式，即:

$$\mathbb{C}^{N/2} \xrightarrow{SIMD} \mathbb{Z}[x]/(x^N+1)$$

明确一下 CKKS SIMD 编码的目标，即寻找这样的多项式 $p(x) \in \mathbb{Z}[x]/(x^N+1)$ ，使得：

$$p(\zeta^{2i+1}) \approx \Delta \cdot \mathbf{z}$$

这里 Δ 是调节因子 (scaling factor)，负责调整 CKKS 的计算精度。$\zeta = exp(\frac{2\pi i}{2N})$ 是 2N-th primitive root of unity。注意上式是约等式，意味着 SIMD 编码一定存在精度误差，通过合理设置 Δ 可以让误差控制在很小的范围, 更详细的实例可以参考 [2].

4.1.2 安全向量内积 Rotate 变换 & 逆编码算法

Alg1: 安全向量内积 Rotate 变换

步骤 1.

1. 将数据打包到多项式 $\frac{A(x)}{X^N+1}$。
2. 将数据打包到多项式 $\frac{B(x)}{X^N+1}$。

步骤 2. 使用自己的公钥加密 $A(x)$，并发送给 B。

步骤 3.

1. 在 $Enc(A(x))$ 和 $B(x)$ 上执行同态乘法，得到 $Enc(C(x))/X^N+1$。
2. $C(x)$ 被打包为 $(a_0b_0, a_1b_1, ..., a_nb_n)$。
3. 注意：仅凭 $C(x)$ 不能直接计算内积。

步骤 4. 对 $C(x)$ 执行旋转，得到 n 个密文：

$$\begin{aligned}
&(a_1b_1, a_2b_2, ...a_nb_n, a_0b_0)\\
&\vdots\\
&(a_2b_2, ...a_nb_n, a_0b_0.a_1b_1)\\
&(a_nb_n, a_0b_0, ...a_{n-1}b_{n-1})
\end{aligned}$$

步骤 5. 对上述 n 个密文求和，发送回 A 解密以获得内积。

分析 可以看到 Rotate 变换中，除了向量在密文域中变换会消耗大量的算力，而且对这 n 个密文求和后，结果向量中的每一项都是内积的结果，但是其实我们只想计算一次内积即可，这大大的浪费了算力。为此，可以改用逆向编码算法去改善时间复杂度。

Alg2: 安全向量内积逆编码算法 舍弃 Packing 机制，将明文直接写在多项式系数上。

步骤 1. A 将数据编码为一个多项式

$$P_a = a_0 + a_1X + a_2X^2 + \ldots + a_nX^n$$

步骤 2. B 将数据编码为一个多项式

$$P_b = b_0 - b_1X^{N-1} - b_2X^{N-2} - \ldots - b_nX^{N-n}$$

已知 $X^N = -1 \mod (X^N+1)$

因此，多项式 $P_a \cdot P_b$ 的乘积的常数项正好是内积 $a \cdot b$，无需昂贵的旋转操作，性能大幅提升。但是，这样的逆编码手段会导致密文多项式中暴露明文，所以需要进一步的加密，具体细节在此不多赘述。

5 分工

1. **原理推断** 精读论文、分析优化原理并抽象出代码逻辑，输出伪代码。
2. **代码实现** 参照伪代码对基础运算 NTT、INTT 以实现并行优化。
3. **代码迁移** 修改接口将并行优化后的数论变换算法接入同态加密工程。
4. **工程优化** 对加密编码原理优化，使后续算法可以实现 SIMD。
5. **运行测试** 涉及多处理器测试脚本的编写与代码测试，需收集处理数据，对数据进行可视化处理并分析算法优化效果。

具体分工：孔德嵘 1, 2 蒋佳豪 3，4，5
可能后续会有调整，不同分工之间也存在相互帮助。

参考文献

[1] XinWei Gao. 【后量子密码】美国国家标准技术研究所（nist）选中首批 4 个后量子密码标准算法, 2022. 2024.

[2] hujwei. 全同态加密之 simd 技术, 2022. 2024.

[3] Steven Yue. 初探全同态加密,fhe 的定义与历史回顾. *zhihu*, 2020.

[4] 刘仲. 面向向量处理器的基于 simd 的 fft 并行计算方法, Mar 2011.

[5] 胡灏. 一种基于多线程技术的并行 fft 算法. 2007.