全同态加密理论、生态现状与未来展望

lynndell2010@gmail.com

mutourend2010@gmail.com

2025 年 5 月 1 日

《全同态加密理论、生态现状与未来展望》系列由 lynndell2010@gmail.com 和 mutourend2010@gmail.com 整理原创发布，分为上中下三个系列：

- 全同态加密理论、生态现状与未来展望（上）：介绍全同态加密理论知识。
- 全同态加密理论、生态现状与未来展望（中）1、（中）2：介绍全同态加密四代算法衍化历程。
- 全同态加密理论、生态现状与未来展望（下）：介绍当前全同态加密生态现状及未来展望。

整个系列内容可能存在纰漏，希望能得到大家的反馈，有任何问题欢迎邮件联系 lynndell2010@gmail.com 和 mutourend2010@gmail.com。最后，在不修改该 PDF 的情况下，允许广泛传播。

1 引言

1.1 格基本概念

1.1.1 格定义

定义 1 (格)**.** 设 $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n \in \mathbb{R}^m$ 为一组线性无关的向量。由 $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n$ 生成的格 L 指的是向量 $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n$ 的线性组合构成的向量集合，且其所有的系数均在 $\mathbb{Z}$ 中，即：

$$L = \{a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n : a_1, a_2, \cdots, a_n \in \mathbb{Z}, \mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n \in \mathbb{R}^m\}.$$

任意一组可以生成格的**线性无关的向量**称为格的**基**，格基的**向量个数**称为格的**维度**。

格的行列式：格的行列式 $\det(L)$ 的值定义为格基本体

$$\mathcal{P}(\mathbf{V}) = \left\{\sum_{i=1}^{n} a_i\mathbf{v}_i, 0 \le a_i < 1\right\}$$

的体积，容易证明

$$\det(L) = \mathrm{vol}(\mathcal{P}(\mathbf{V})) = \sqrt{\mathbf{V}^T\mathbf{V}}$$

其中，V 是以格为列构成的矩阵。换言之，一个格 L 中，优质基与劣质基的行列式值相等。

1.1.2 优质基与劣质基

- **优质基：**向量角度比较大（Hadamard 比率较大，接近 1）。使用优质基，调用 Babai 算法，**能**在多项式时间内求解 CVP 困难问题；
- **劣质基：**向量角度比较小（Hadamard 比率较小，接近 0）。使用劣质基，**不能**在多项式时间内求解 CVP 困难问题。

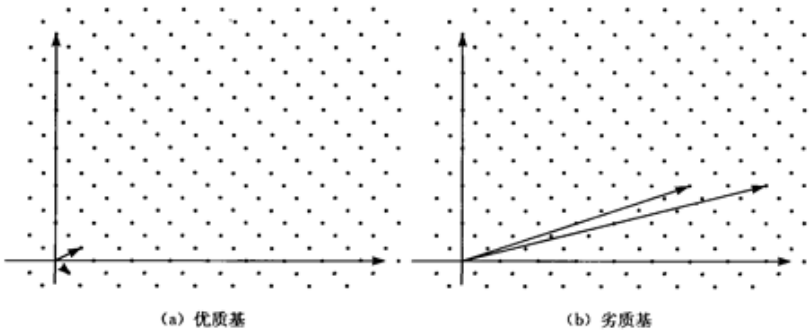

图 1: 优质基与劣质基

已知优质基 $\mathbf{V} = (\mathbf{v}_1, ..., \mathbf{v}_n)$，能在多项式时间内求劣质基 $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_n)$。

求解方法：选择随机矩阵 $\mathbf{A}$，要求 $\det(\mathbf{A}) = \pm 1$，则

$$\mathbf{W} := \mathbf{AV}$$

反之，已知劣质基 $\mathbf{W}$，**不能**在多项式时间内求优质基 $\mathbf{V}$。

1.1.3 格计算困难问题

- **最短向量问题 (SVP):** 在格 L 中求一个非零向量 $\mathbf{v} \in L$，使它的欧几里得范数 $||\mathbf{v}|| = \sqrt{\mathbf{v} \cdot \mathbf{v}}$ 最小。
- **最近向量问题 (CVP):** 已知一个不在格 L 中的向量 $\mathbf{w} \in \mathbb{R}^m$，求一个向量 $\mathbf{v} \in L$，使它最接近 $\mathbf{w}$。换言之，求一个向量 $\mathbf{v} \in L$，使欧几里得范数 $||\mathbf{w} - \mathbf{v}||$ 最小。

简单理解如下：

- **CVP 问题困难性：**已知劣质基，需要使用劣质基枚举所有格点；然后计算每个格点与目标点的距离；最后筛选法找出最短距离。
- **SVP 问题困难性：**已知劣质基，求优质基；然后枚举零点附近的格点；最后找到最短向量。

在格中可能存在不止一个最短的非零向量（找出一个即可）。例如，在 $\mathbb{Z}^2$ 中，$(0, \pm 1)$ 和 $(\pm 1, 0)$ 这四个向量都是 SVP 解。这种情形也适用于 CVP。

随着格的维度 n 的增加，在计算上也越来越难解。另外，即使是对 SVP 和 CVP 的近似解，在理论和应用数学的诸多领域都有许多应用。通常，CVP 被认为是 NP 难问题，SVP 在特定的“随机规约假设”下也被认为是 NP 难问题。

1.1.4 最短基问题

在实际应用中，有一些很重要的 SVP 和 CVP 的变形最短基问题 SBP：求一个格基 $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$，使它在某些情况下最短。如可能需要使下式最小：

$$max_{1 \le i \le n} ||\mathbf{v}_i|| \quad \vee \quad \sum_{i=1}^{n} ||\mathbf{v}_i||^2$$

因此，可能有许多版本的 SBP，这取决于如何定义基的“大小”。

1.1.5 近似最短/最近向量问题

近似最短向量问题 (apprSVP)：设 $\Psi(n)$ 是 n 的一个函数。在 n 维格 L 中，求一个非零向量，使其不大于最短非零向量的 $\Psi(n)$ 倍。换言之，如果 $\mathbf{v}_{\text{shortest}}$ 是格 L 的最短非零向量，则求一个非零向量 $\mathbf{v} \in L$，使其满足：

$$||\mathbf{v}|| \leq \Psi(n)||\mathbf{v}_{\text{shortest}}||$$

不同函数 $\Psi(n)$ 可形成不同的 apprSVP。如要求设计一个算法，用于求非零的向量 $\mathbf{v} \in L$，满足：

$$||\mathbf{v}|| \leq 3\sqrt{n}||\mathbf{v}_{\text{shortest}}|| \quad \vee \quad ||\mathbf{v}|| \leq 2^{n/2}||\mathbf{v}_{\text{shortest}}||$$

很明显，前者的算法比后者要强，但如果格的维度不大，即使对后者的解也可能是非常有用的。

近似最近向量问题 (apprCVP)：与 apprSVP 类似，需要求 CVP 的近似解。

1.1.6 LWE 困难问题

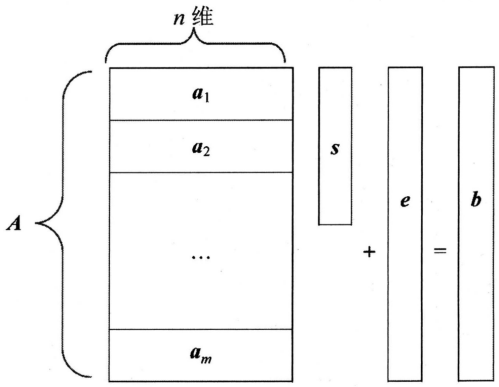

图 2: LWE 搜索困难问题

- **LWE 搜索困难问题：**已知矩阵 $\mathbf{A} \in \mathbb{Z}_q^{mn}$ 和向量 $\mathbf{b} \in \mathbb{Z}_q^m$，求向量 $\mathbf{s} \in \mathbb{Z}_q^n$ 是困难的。其中，

$$\mathbf{b} = \mathbf{As} + \mathbf{e} \mod q.$$

  $\mathbf{e} \in \mathbb{Z}_N^m$ 是噪声（欧几里得范数很小）。
- **LWE 判决困难问题：**已知矩阵 $\mathbf{A} \in \mathbb{Z}_q^{mn}$ 和向量 $\mathbf{b} \in \mathbb{Z}_q^m$，判断是 LWE 实例，还是随机数实例。

Regev 在 2005 年研究了 LWE 问题的困难性，证明在量子归约下，LWE 至少与 worst-case 的近似因子为 $\tilde{O}(n/\alpha)$ 的 SVP 的变体一样困难。其中，是 LWE 实例中与扰动分布的方差有关的参数。On lattices, learning with errors, random linear codes, and cryptography

1.1.7 环 LWE 困难问题

模 $q \geq 2$，多项式次数 $n \geq 1$ 是 2 的幂次方，多项式为 $f(x) = x^n + 1$。环 $R = \mathbb{Z}[x]/f(x)$, 环 $R_q = \mathbb{Z}_q[x]/f(x)$。$\chi$ 是 R 上的一个噪声概率分布。$\bar{A}_{s,\chi}$ 是 $R_q \times R_q$ 上的一个概率分布。该分布以如下方式获得：随机选择 $\mathbf{a} \in R_q$，根据分布 χ 选择噪声向量 $\mathbf{e} \in R_q$，则

$$\bar{A}_{s,\chi} = (\mathbf{a}, \mathbf{b}) = (\mathbf{a}, \mathbf{a} \bullet \mathbf{s} + \mathbf{e}) \in R_q \times R_q$$

- **环 LWE 搜索困难问题：**已知 $(\mathbf{a},\mathbf{b}) \in R_q \times R_q$，求 $\mathbf{s} \in R_q$ 是困难的。
- **环 LWE 判决困难问题：**已知环 LEW$(\mathbf{a},\mathbf{b}) \in R_q \times R_q$ 实例与随机均匀分布实例 $(\mathbf{a}',\mathbf{b}')$，无法区分。

1.1.8 Babai 算法

已知优质基 $\mathbf{V}=(\mathbf{v}_1,...,\mathbf{v}_n)$，求**目标向量 w** 的**最近向量 w′**，即解决 CVP 困难问题。

Babai 算法 $\mathbf{w}'=\lceil \mathbf{w}\mathbf{V}^{-1} \rfloor \mathbf{V}$。具体而言，包括以下三个步骤：

1. 使用优质基，能在多项式时间内解方程组，求出实数 $t_i \in \mathbb{R}$，使得 $\mathbf{w}=t_1\mathbf{v}_1+...+t_n\mathbf{v}_n$，即任意向量由基线性表达。
2. 对于实数 $t_i \in \mathbb{R}$，取四舍五入【去噪声】$a_i=\lceil t_i \rfloor$。
3. 计算向量 $\mathbf{w}':=a_1\mathbf{v}_1+...+a_n\mathbf{v}_n$，则 $\mathbf{w}'$ 是 **w** 的**最近向量**。

举例：

- 已知最优质基 $\begin{pmatrix}10\\0\end{pmatrix}\begin{pmatrix}0\\10\end{pmatrix}$ 和目标向量 $w=\begin{pmatrix}100\\100\end{pmatrix}$ 求出 $t_1=t_2=10$，则 $a_1=a_2=10$【无误差】，则最近向量 $w'=\begin{pmatrix}100\\100\end{pmatrix}$，**是目标向量**。
- 已知优质基 $\begin{pmatrix}101\\0\end{pmatrix}\begin{pmatrix}0\\101\end{pmatrix}$ 和目标向量 $w=\begin{pmatrix}100\\100\end{pmatrix}$ 求出 $t_1=t_2=100/101=0.99$，则 $a_1=a_2=1$【误差小】，则最近向量 $w'=\begin{pmatrix}101\\101\end{pmatrix}$，**接近目标向量**。
- 已知劣质基 $\begin{pmatrix}101\\201\end{pmatrix}\begin{pmatrix}201\\101\end{pmatrix}$ 和目标向量 $w=\begin{pmatrix}100\\100\end{pmatrix}$ 求出 $t_1=t_2=50/151=0.33$，则 $a_1=a_2=0$【误差大】，则最近向量 $w'=\begin{pmatrix}0\\0\end{pmatrix}$，**远离目标向量**。

分析：

使用**优质基** $(\mathbf{v}_1,...,\mathbf{v}_n)$，使得对 t_i 四舍五入时，误差较小，所以最终结果正确。

使用**劣质基** $(\mathbf{w}_1,...,\mathbf{w}_n)$ 在多项式时间内求实数 $t_i \in \mathbb{R}$，则四舍五入的误差较大，无法获得正确结果。

如果目标向量 w 是一个**噪声**（欧几里得范数很小），则 t_i 非常接近 0，四舍五入后获得的 a_i 等于 0。

- 已知最优质基 $\begin{pmatrix}10\\0\end{pmatrix}\begin{pmatrix}0\\10\end{pmatrix}$ 和目标向量是**噪声** $w=\begin{pmatrix}2\\-3\end{pmatrix}$ 求出 $t_1=0.2,t_2=-0.3$，则 $a_1=a_2=0$，则最近向量 $w'=\begin{pmatrix}0\\0\end{pmatrix}$，**去噪声**。
- 已知最优质基 $\begin{pmatrix}10\\0\end{pmatrix}\begin{pmatrix}0\\10\end{pmatrix}$ 和目标向量是格点 + **噪声** $w=\begin{pmatrix}12\\-13\end{pmatrix}$ 求出 $t_1=1.2,t_2=-1.3$，则 $a_1=1,a_2=-1$，则最近向量 $w'=\begin{pmatrix}10\\-10\end{pmatrix}$，**去噪声，找到目标向量**。

因此，Babai 算法输入优质基，去噪声，误差小，能找到最近向量，解决 CVP 困难问题。

反之，Babai 算法输入劣质基，去噪声，误差大，不能找到最近向量，不能解决 CVP 困难问题。

1.1.9 GGH 公钥密码系统

Public-key cryptosystems from lattice reduction problems

- **密钥生成：**选择一个优质基 $\mathbf{V}=(\mathbf{v}_1,...,\mathbf{v}_n)$ 作为私钥。选择一个整数矩阵 $\mathbf{U}$，使得 $\det(\mathbf{U})=\pm1$。计算 $\mathbf{W}:=\mathbf{UV}$，则获得劣质基 $\mathbf{W}=(\mathbf{w}_1,...,\mathbf{w}_n)$ 作为公钥。
- **加密：**消息为小向量 $\mathbf{m}$，选择噪声 $\mathbf{r}$，输入公钥 $\mathbf{W}$，如下计算：

$$\mathbf{c}=\mathbf{m}\cdot\mathbf{W}+\mathbf{r}$$

  则密文为 $\mathbf{c}$。
- **解密：**调用 Babai 算法，输入私钥 (优质基)$\mathbf{V}$，如下解密：

$$\begin{aligned}
&\lceil\mathbf{cV}^{-1}\rfloor\bullet\mathbf{VW}^{-1}\\
&=\lceil(\mathbf{m}\cdot\mathbf{W}+\mathbf{r})\mathbf{V}^{-1}\rfloor\bullet\mathbf{U}^{-1}\\
&=\lceil\mathbf{m}\cdot\mathbf{WV}^{-1}+\mathbf{rV}^{-1}\rfloor\bullet\mathbf{U}^{-1}\\
&=\lceil\mathbf{m}\cdot\mathbf{UVV}^{-1}+\mathbf{rV}^{-1}\rfloor\bullet\mathbf{U}^{-1}\\
&=\lceil\mathbf{m}\cdot\mathbf{U}+\mathbf{rV}^{-1}\rfloor\bullet\mathbf{U}^{-1}\\
&\approx\mathbf{m}\cdot\mathbf{U}\bullet\mathbf{U}^{-1}\\
&=\mathbf{m}
\end{aligned}$$

Babai 算法中，如果噪声 $\mathbf{r}$ 欧几里得范数很小，则步骤 1 的 t_i 一定更小，四舍五入后就是 0，所以 $\lceil\mathbf{rV}^{-1}\rfloor\approx0$。

举例：

- **密钥生成：**私钥使用如下**优质基**：

$$\mathbf{V}=\begin{bmatrix}\mathbf{v}_1\\\mathbf{v}_2\\\mathbf{v}_3\\\mathbf{v}_4\\\mathbf{v}_5\end{bmatrix}=\begin{bmatrix}81&15&17&60&29\\-53&7&49&46&-11\\2&84&-6&-68&-97\\11&-96&92&70&-70\\28&-58&98&-89&24\end{bmatrix}$$

  由 $v_1,v_2,\ldots,v_5$ 作为基构成的格 L 的行列式值为 $\det(L)=22655546896$ 该基的 Hadamard 比率为

$$\mathcal{H}(v_1,v_2,\ldots,v_5)=\left[\frac{\det(L)}{\|v_1\|\|v_2\|\|v_3\|\|v_4\|\|v_5\|}\right]^{1/5}\approx0.9249$$

  接近 1，所以是**优质基**。

选择如下随机矩阵：

$$\mathbf{U} = \begin{bmatrix} 16 & 111 & 139 & -16 & -95 \\ -91 & -642 & -747 & 185 & 471 \\ -103 & -677 & -1133 & 492 & 524 \\ -21 & -145 & -190 & 55 & 111 \\ -10 & 86 & 9 & -82 & 62 \end{bmatrix}$$

满足 $\det(\mathbf{U}) = 1$。然后与私钥相乘，得到**劣质基**作为公钥：

$$\mathbf{W} = \mathbf{UV} = \begin{bmatrix} -7145 & 19739 & -4237 & 3949 & -15400 \\ 40384 & -113685 & 25691 & -13165 & 75236 \\ 45356 & -179080 & 54894 & 27526 & 92497 \\ 9137 & -29008 & 7336 & -1039 & 18230 \\ 4600 & 4280 & -5798 & -16426 & 7011 \end{bmatrix}$$

可以看出，这组公钥基的 Hadamard 比率相对于私钥基来说非常小：

$$\mathcal{H}(w_1, w_2, \ldots, w_5) = \left[\frac{\det(L)}{\|w_1\|\|w_2\|\|w_3\|\|w_4\|\|w_5\|} \right]^{1/5} \approx 0.0021$$

接近 0，所以是**劣质基**。

- **加密：**消息为 $\mathbf{m} = (-78, 48, 5, 66, 89)$，选择随机噪声 $\mathbf{r} = (-9, -5, 1, -2, 4)$，使用如下公式加密：

$$\mathbf{c} = \mathbf{mW} + \mathbf{r}$$

计算得到：

$$\mathbf{c} = \begin{bmatrix} -78 & 48 & 5 & 66 & 89 \end{bmatrix} \cdot \begin{bmatrix} -7145 & 19739 & -4237 & 3949 & -15400 \\ 40384 & -113685 & 25691 & -13165 & 75236 \\ 45356 & -179080 & 54894 & 27526 & 92497 \\ 9137 & -29008 & 7336 & -1039 & 18230 \\ 4600 & 4280 & -5798 & -16426 & 7011 \end{bmatrix} + \begin{bmatrix} -9 \\ -5 \\ 1 \\ -2 \\ 4 \end{bmatrix}$$

$$= (3746835, -9425535, 1806279, -2332802, 7102176)$$

- **解密：**调用 Babai 算法解密，将 $\mathbf{c}$ 用私钥（优质基）$v_1, v_2, \ldots, v_5$ 表达：由于 $\mathbf{c}$ 中含有噪声 $\mathbf{r}$，因此有：

$$\mathbf{c} \cdot \mathbf{V}^{-1} \approx \begin{bmatrix} -8407.083 \\ -60082.952 \\ -64102.054 \\ 8919.983 \\ 45482.020 \end{bmatrix}$$

四舍五入，再与私钥（优质基）$\mathbf{V}$ 相乘，即得到一个最近格向量（解决了 CVP 问题）：

$$\begin{aligned} \mathbf{v}' &= \lceil \mathbf{c} \cdot \mathbf{V}^{-1} \rfloor \cdot \mathbf{V} \\ &= \begin{bmatrix} -8407, -60083, -64102, 8920, 45482 \end{bmatrix} \cdot \mathbf{V} \\ &= \begin{bmatrix} 3746844, -9425530, 1806278, -2332800, 7102172 \end{bmatrix} \end{aligned}$$

这个结果等于前面的 mW，即密文 c 去掉了噪声。即 $\mathbf{v}' = \mathbf{c} - \mathbf{r} = \mathbf{mW}$。接下来计算：

$$\mathbf{m} = \mathbf{v}' \cdot \mathbf{W}^{-1} = \Big[3746844, -9425530, 1806278, -2332800, 7102172\Big] \cdot W^{-1} = \Big[-78, 48, 5, 66, 89\Big]$$

解密成功。

- **攻击：**假设攻击者试图去破解密文，但只知道公钥（劣质基）**W**。如果它对于公钥 **W** 应用 Babai 算法的话，会得到：

$$\mathbf{c} \cdot \mathbf{W}^{-1} \approx \Big[3417.187, 5205.909, -62877.902, 351125.565, -130786.869\Big]$$

然后四舍五入，同样会得到一个格向量:

$$\mathbf{v}'' = \lceil \mathbf{c} \cdot \mathbf{W}^{-1} \rfloor = \Big[3417, 5206, -62878, 351126, -130787\Big]$$

然后与公钥 **W** 相乘，即 $\mathbf{v}'' \cdot \mathbf{W} = \lceil \mathbf{c} \cdot \mathbf{W}^{-1} \rfloor \cdot \mathbf{W}$。很明显，攻击者找到的是一个不正确的明文向量:

$$m = \Big[3417, 5206, -62878, 351126, -130787\Big]$$

对于不同的基，使用 Babai 算法的效果如下：

$$\|\mathbf{c} - \mathbf{v}'\| = \|\mathbf{c} - \lceil \mathbf{cV}^{-1} \rfloor \bullet \mathbf{V}\| = 11.2694$$
$$\|\mathbf{c} - \mathbf{v}''\| = \|\mathbf{c} - \lceil \mathbf{cW}^{-1} \rfloor \bullet \mathbf{W}\| \approx 13264.50$$

因此，对于劣质基 **W**，Babai 算法的结果不能作为 CVP 的解。

1.2 典型加密方案

1.2.1 Regev05：加密 1 比特

On Lattices, Learning with Errors, Random Linear Codes, and Cryptography

格的维数为 n，χ 是 $\mathbb{Z}$ 上的噪声高斯分布，其欧几里得范数较小。

- **密钥生成：**私钥为随机向量 $sk = \mathbf{s} \leftarrow \mathbb{Z}_q^n$。随机均匀选取矩阵 $\mathbf{A} \leftarrow \mathbb{Z}_q^{N \times n}$ 和噪声 $\mathbf{e} \leftarrow \chi^N$，如下计算

$$\mathbf{b} := \mathbf{As} + \mathbf{e}$$

则公钥为 $PK = (\mathbf{A}, \mathbf{b})$。

- **加密：**对于消息 $m \in \{0, 1\}$，选择随机数 $\mathbf{r} \in \{0, 1\}^N$，如下计算

$$c_1 := \left\lfloor \frac{q}{2} \right\rfloor \cdot m + \mathbf{r}^T \cdot \mathbf{b} \in \mathbb{Z}_q$$
$$\mathbf{c}_2 := \mathbf{r}^T \mathbf{A} \in \mathbb{Z}_q^n$$

则密文为 $(c_1, \mathbf{c}_2)$。

- **解密：**输入私钥 **s** 和密文 $(c_1, \mathbf{c}_2)$，如下计算

$$\left\lfloor \frac{2}{q} [(c_1 - \mathbf{c}_2 \cdot \mathbf{s}) \mod q] \right\rfloor \mod 2$$

展开如下

$$\begin{aligned}&\left\lfloor \frac{2}{q}\left[(c_1-\mathbf{c}_2\cdot\mathbf{s}) \mod q\right]\right\rfloor\\&=\frac{2}{q}\left[\left\lfloor\frac{q}{2}\right\rfloor\cdot m+\mathbf{r}^T\cdot\mathbf{b}-\mathbf{r}^T\mathbf{A}\mathbf{s}\right]\\&=\frac{2}{q}\left[\left\lfloor\frac{q}{2}\right\rfloor\cdot m+\mathbf{r}^T\cdot(\mathbf{A}\mathbf{s}+\mathbf{e})-\mathbf{r}^T\mathbf{A}\mathbf{s}\right]\\&=\frac{2}{q}\left[\left\lfloor\frac{q}{2}\right\rfloor\cdot m+\mathbf{r}^T\cdot\mathbf{e}\right]\\&=m+e^*\end{aligned}$$

当噪声 e^* 小于 $\left\lfloor\frac{q}{2}\right\rfloor/2$ 时，解密是精确的。注意，噪声 $\mathbf{r}^T\cdot\mathbf{e}$ 被放大了。

1.2.2 BV11：加密 1 比特

Efficient Fully Homomorphic Encryption from (Standard) LWE

- **密钥生成：**私钥为随机向量 $sk=\mathbf{s}\leftarrow\mathbb{Z}_q^n$。随机均匀选取矩阵 $\mathbf{A}\leftarrow\mathbb{Z}_q^{N\times n}$ 和噪声 $\mathbf{e}\leftarrow\chi^N$，如下计算

$$\mathbf{b}:=\mathbf{A}\mathbf{s}+2\mathbf{e}$$

则公钥为 $PK=(\mathbf{A},\mathbf{b})$。注意：噪声为偶数倍。

- **加密：**对于消息 $m\in\{0,1\}$，选择随机数 $\mathbf{r}\in\{0,1\}^N$，如下计算

$$\begin{aligned}c_1&:=m+\mathbf{b}^T\cdot\mathbf{r}\in\mathbb{Z}_q\\\mathbf{c}_2&:=\mathbf{A}^T\mathbf{r}\in\mathbb{Z}_q^n\end{aligned}$$

- **解密：**输入私钥 $\mathbf{s}$ 和密文 $(c_1,\mathbf{c}_2)$，如下计算

$$(c_1-\mathbf{c}_2\cdot\mathbf{s}) \mod q \mod 2$$

展开如下

$$\begin{aligned}&(c_1-\mathbf{c}_2\cdot\mathbf{s}) \mod q \mod 2\\&=(m+\mathbf{b}^T\cdot\mathbf{r}-\mathbf{A}^T\mathbf{r}\cdot\mathbf{s}) \mod q \mod 2\\&=(m+(\mathbf{A}\mathbf{s}+2\mathbf{e})^T\cdot\mathbf{r}-\mathbf{A}^T\mathbf{r}\cdot\mathbf{s}) \mod q \mod 2\\&=m+2\mathbf{e}^T\mathbf{r} \mod 2\end{aligned}$$

注意，噪声 $2\mathbf{e}^T\mathbf{r}$ 被放大了。

1.2.3 KTX07：加密 l 比特

Lattice-based cryptography

- **密钥生成：**选择随机矩阵 $\mathbf{S}'\leftarrow\mathbb{Z}_q^{n\times l}$，则私钥为 $sk=\mathbf{S}=(1,...,1|\mathbf{S}')\in\mathbb{Z}_q^{(n+l)\times l}$。其中，$l$ 维单位阵与矩阵 $\mathbf{S}'$ 拼接。

  随机均匀选取矩阵 $\mathbf{A}'\leftarrow\mathbb{Z}_q^{N\times n}$ 和**噪声矩阵** $\mathbf{E}\leftarrow\chi^{N\times l}$，如下计算

$$\mathbf{b}:=\mathbf{A}'\mathbf{S}'+\mathbf{E}\in\mathbb{Z}_q^{N\times l}$$

令

$$\mathbf{A} := [\mathbf{b}| - \mathbf{A}'] \in \mathbb{Z}_q^{N\times(n+l)}$$

则公钥为 $PK = \mathbf{A}$。以下等式成立：

$$\mathbf{A}\cdot\mathbf{S} = [\mathbf{b}| - \mathbf{A}']\cdot\mathbf{S} = [\mathbf{A}'\mathbf{S}' + \mathbf{E}| - \mathbf{A}']\cdot\mathbf{S} = [\mathbf{A}'\mathbf{S}' + \mathbf{E}| - \mathbf{A}']\cdot(1,...,1|\mathbf{S}') = \mathbf{E}\ (This\ is\ noise.)$$

- **加密：** 消息为 l 维向量 $\mathbf{m}\in\mathbb{Z}_t^l$，令 $\mathbf{m}' = (\mathbf{m}, 0, ..., 0)\in\{0,1\}^{n+l}$。选择随机向量 $\mathbf{r}\in\{-a, -a+1, ..., a\}^N$

$$\mathbf{c} := \left\lceil (\frac{q}{t})\mathbf{m}' \right\rfloor + \mathbf{A}^T\cdot\mathbf{r} \in \mathbb{Z}_q^{(n+l)}$$

则密文为 $\mathbf{c}$。

- **解密：** 输入私钥 $\mathbf{S}$ 和密文 $\mathbf{c}$，如下解密

$$\mathbf{m} := \left\lceil \frac{t}{q}(\mathbf{S}^T\mathbf{c}) \right\rfloor$$

展开如下

$$\left\lceil \frac{t}{q}(\mathbf{S}^T\mathbf{c}) \right\rfloor = \left\lceil \frac{t}{q}\left(\mathbf{S}^T\left(\left\lceil (\frac{q}{t})\mathbf{m}' \right\rfloor + \mathbf{A}^T\cdot\mathbf{r}\right)\right) \right\rfloor = \left\lceil \frac{t}{q}\left(((1,...,1|\mathbf{S}'))^T\left\lceil (\frac{q}{t})((\mathbf{m},0,...,0)) \right\rfloor + \mathbf{E}^T\cdot\mathbf{r}\right) \right\rfloor$$

注意，噪声 $\frac{t}{q}\mathbf{E}^T\mathbf{r}$ 被放大了。

1.2.4 DGHV10：基于整数的 FHE

Fully homomorphic encryption over the integers

- **密钥生成：** 私钥为随机整数 p。公钥为 pq。q 是另外一个随机整数。
- **加密：** 对于消息 $m\in\{0,1\}$，选择噪声 e，计算

$$c := m + 2e + pq$$

则密文为 c。

- **解密：** 输入私钥 p 和密文 c，计算

$$(c \mod p) \mod 2.$$

模 p 去掉了 pq，模 2 去掉了噪声，留下消息 m。

- **同态加法：** $c + c' = (m + m') + 2(e + e') + p(q + q')$
- **解密：** $(c + c' \mod p) \mod 2 = m + m'$。加法对噪声影响较小
- **同态乘法：** $c\cdot c' = (m + 2e)(m' + 2e') + p(pqq' + mq' + m'q + 2eq' + 2e'q)$
- **解密：** $(c\cdot c' \mod p) \mod 2 = (m + 2e)(m' + 2e') \mod p \mod 2$。

关键项 $(m + 2e)(m' + 2e')$ 乘积后变大，可能超过 p，则 mod p 有问题，导致解密出错。

1.2.5 LV10：环 LEW 公钥加密

- 多项式的模运算非常适合图灵计算架构（分配的内存宽度有限）。
- 矩阵计算需要根据计算需求额外分配内存，所以效率稍微差点。

LV10 环 LEW 公钥加密看作 Regev 的 LEW 公钥加密在环上的推广。

On Ideal Lattices and Learning with Errors over Rings

模 $q \geq 2$，多项式次数 $n \geq 1$ 是 2 的幂次方，多项式为 $f(x) = x^n + 1$。环 $R = \mathbb{Z}[x]/f(x)$, 环 $R_q = \mathbb{Z}_q[x]/f(x)$。$\chi$ 是 R 上的一个噪声概率分布。

- **密钥生成：** 随机选择 $\mathbf{s} \in \chi$，作为私钥。随机选择 $\mathbf{a} \in R_q$，选择噪声 $e_1 \in \chi$，计算

$$\mathbf{b} = \mathbf{as} + e_1$$

  则公钥为 $(\mathbf{b}, \mathbf{a}) \in R_q \times R_q$。

- **加密：** 消息为 $\mathbf{m} \in \{0,1\}^n$，其多项式系数 $m \in R_2$。随机选择 $e_2, e_3, e_4 \in \chi$，如下计算

$$c_1 = \lfloor q/2 \rfloor \cdot m + \mathbf{b}e_2 + e_3 \in R_q$$

$$c_2 = \mathbf{a}e_2 + e_4 \in R_q$$

  则密文为 $\mathbf{c} = (c_1, c_2)$。

- **解密：** 输入私钥 $\mathbf{s}$ 和密文 $\mathbf{c}$，如下解密

$$m := \frac{2}{q}\lfloor c_1 - c_2\mathbf{s} \mod q \rceil \mod 2$$

  展开如下：

$$\begin{aligned}
&\frac{2}{q}\lfloor c_1 - c_2\mathbf{s} \rceil \\
&= \frac{2}{q}(\lfloor q/2 \rfloor \cdot m + \mathbf{b}e_2 + e_3 - (\mathbf{a}e_2 + e_4)\mathbf{s}) \\
&= \frac{2}{q}(\lfloor q/2 \rfloor \cdot m + (\mathbf{as} + e_1)e_2 + e_3 - (\mathbf{a}e_2 + e_4)\mathbf{s}) \\
&= \frac{2}{q}(\lfloor q/2 \rfloor \cdot m + e_1e_2 + e_3 - e_4\mathbf{s}) \\
&= m + \frac{2}{q}(e_1e_2 + e_3 - e_4\mathbf{s}) \\
&= m + e^*
\end{aligned}$$

1.2.6 LV10 变形

方案特点：2 倍噪声，则加密不需要放大消息，解密不需要缩小消息。

模 $q \geq 2$，多项式次数 $n \geq 1$ 是 2 的幂次方，多项式为 $f(x) = x^n + 1$。环 $R = \mathbb{Z}[x]/f(x)$, 环 $R_q = \mathbb{Z}_q[x]/f(x)$。$\chi$ 是 R 上的一个噪声概率分布。

- **密钥生成：** 随机选择 $\mathbf{s} \in \chi$，作为私钥。随机选择 $\mathbf{a} \in R_q$，选择 $e_1 \in \chi$，计算

$$\mathbf{b} = \mathbf{as} + 2e_1$$

  则公钥为 $(\mathbf{b}, \mathbf{a})$。

- **加密：** 消息为 $\mathbf{m} \in \{0,1\}^n$，其多项式系数 $m \in R_2$。随机选择 $e_2, e_3, e_4 \in \chi$，如下计算

$$c_1 = \mathbf{m} + \mathbf{b}e_2 + e_3 \in R_q$$

$$c_2 = \mathbf{a}e_2 + 2e_4$$

  则密文为 $\mathbf{c} = (c_1, c_2)$。

- **解密：**输入私钥 **s** 和密文 **c**，如下解密

$$\mathbf{m} := c_1 - c_2\mathbf{s} \mod q \mod 2$$

展开如下：

$$\begin{aligned}
&c_1 - c_2\mathbf{s} \mod q \mod 2 \\
&= \mathbf{m} + \mathbf{b}e_2 + 2e_3 - (\mathbf{a}e_2 + 2e_4)\mathbf{s}) \mod q \mod 2 \\
&= \mathbf{m} + (\mathbf{a}\mathbf{s} + 2e_1)e_2 + 2e_3 - (\mathbf{a}e_2 + 2e_4)\mathbf{s}) \mod q \mod 2 \\
&= \mathbf{m} + 2e_1e_2 + 2e_3 - 2e_4\mathbf{s} \mod q \mod 2 \\
&= \mathbf{m} + 2e^* \mod 2 \\
&= \mathbf{m}
\end{aligned}$$

1.3 FHE 基本概念

全同态加密（Fully Homomorphic Encryption, FHE）是一种加密方案，它允许在密文上直接进行运算，从而在无需解密的情况下实现对明文的任意操作。这种功能的实现仅在加法和乘法操作可以同态执行的条件下才可能，因为这两种操作在有限环上构成一个功能完备的集合。

具体而言，任何布尔（或算术）电路都可以仅通过 XOR（加法）和 AND（乘法）门表示。换句话说，给定两个密文 Enc(x) 和 Enc(y)（其中，x 和 y 是明文，Enc 是加密操作），可以通过直接对密文进行加法（或乘法）运算，得到 $x + y$（或 $x \cdot y$）的加密结果，而无需解密 Enc(x) 和 Enc(y)。这足以对加密数据进行任何函数的评估，满足以下性质：Dec(Enc(x) Δ Enc(y)) $= x \,\Delta\, y$, 其中，Δ 表示加法或乘法，Dec 是解密操作。

全同态加密方案由一组概率多项式时间（PPT）算法 (KeyGen, Enc, Dec, Eval) 组成，满足以下性质：

- **密钥生成 (KeyGen)：**该算法以安全参数 λ 作为输入，输出密钥对 (sk, pk) 和用于在密文上执行同态操作的评估密钥 evk 。
- **加密 (Enc)：**该算法以公钥 pk 和消息 m（来自消息空间）为输入，输出密文 c 。
- **解密 (Dec)：**该算法以私钥 sk 和密文 c 为输入，输出消息 m 。如果解密算法无法成功恢复加密的消息 m，则输出 $\perp$ 。
- **评估 (Eval)：**该算法以评估密钥 evk 、一个函数 f 和一组密文 $(c_1, \ldots, c_t)$ 为输入，输出密文 c_f 。解密 c_f 后得到 $f(m_1, \ldots, m_t)$ 的结果，即：$c_f = \text{Eval}_{\text{evk}}(f, (c_1, \ldots, c_t))$，$\text{Dec}_{\text{sk}}(c_f) = f(m_1, \ldots, m_t)$。注意，密文 c_f 和 $c \leftarrow \text{Enc}_{\text{pk}}(f(m_1, \ldots, m_t))$ 在解密后得到相同的明文，但其构造方式可能不同（如噪声水平可能不同）。

同态加密方案的两个基本特性为：

- **所支持函数的最大 degree：**该特性定义了方案 E 能正确评估的函数范围。具体而言，若 E 能正确评估函数集合 F 中的任何函数 f，即存在评估算法 Eval，满足：

$$\text{Dec}_{\text{sk}}(\text{Eval}_{\text{evk}}(f, c_1, \ldots, c_t)) = f(m_1, \ldots, m_d), \quad \forall f \in F,$$

其中，$c_i \leftarrow \text{Enc}_{\text{pk}}(m_i), \forall i \in \{1, \ldots, t\}$ 。此外，该特性决定了评估密文 c_f（即 Eval 的输出）是否能用作后续评估算法的输入。在多跳同态加密方案中，可以对加密消息 m 生成的密文 c 逐一进行一系列多项式 degree 的函数同态评估。

- **密文长度的增长：**该特性描述了每次评估后密文 bit 长度的增长情况。如果密文长度增长的上限与函数 f 的复杂度无关，则称该方案为紧凑型方案。

根据上述特性，同态加密方案可分为以下几类：

- **Partially HE (PHE)** 只能评估包含一种算术门（即加法或乘法）的电路，且电路深度不受限制的方案。
- **Leveled HE (LHE)** 能够评估包含加法和乘法门的电路，但仅限于预定的乘法深度 L 的方案。
- **Somewhat HE (SHE)** 能够评估包含加法和乘法门的电路子集，其复杂度随着电路深度增加而增加。SHE 比 LHE 更为通用。示例包括 Gentry（2009, 2010）。
- **Leveled Fully HE** 几乎与有级同态加密相同，但这些方案能够评估深度为 L 的所有电路。示例包括 Brakerski 和 Vaikuntanathan（2014）；Brakerski 等（2014）；Brakerski（2012）。
- **Fully HE (FHE)** 能够评估包含加法和乘法门的所有电路，且电路深度不受限制的方案。示例包括 Gentry（2009）和 Brakerski 及 Vaikuntanathan（2014）；Brakerski 等（2014）；Brakerski（2012），在弱环形安全模型下，这种模型保证在只使用一对私钥和公钥时的安全性。

根据FHE 方案如何建模计算，FHE 方案可以分为 3 种类型：

- 将计算建模为 Boolean Circuit 布尔电路（即 bits 位），如 TFHE 方案。
- 将计算建模为 Modular Arithmetic 模运算（即“clock”运算），如 BGV 和 BFV 方案。
- 将计算建模为 Floating Point arithmetic 浮点算法，如 CKKS 方案。

值得强调的是，具有足够同态评估能力的部分同态加密方案（或分层全同态加密方案）可以通过“引导”技术转化为全同态加密方案。

FHE（Fully Homomorphic Encryption，全同态加密）在不解密的情况下，能够对密文执行任意计算。计算后的密文结果解密后，等于对明文做同样计算的结果。即，对于任意有效的函数 f 和明文 m，FHE 具备的性质为：$f(Enc(m)) = Enc(f(m))$。

定义 2 (全同态加密). *全同态加密算法包含 4 个概率多项式时间算法：密钥生成，加密，解密，密文计算。*

- **密钥生成 *KeyGen*** (1^λ)**：** *输出公钥 pk，计算公钥 evk 和私钥 sk。*
- **加密 *Enc*** (pk, m)**：** *使用公钥 pk 加密一位消息 $m \in \{0,1\}$，输出密文 c。*
- **解密 *Dec*** (sk, c)**：** *使用密钥 sk 解密密文 c，恢复消息 m。*
- **同态计算 *Eval*** $(evk, f, c_1, \cdots, c_t)$**：** *使用计算公钥 evk，将 $c_1, c_2, \cdots, c_t$ 输入函数 f。其中，$f : \{0,1\}^* \to \{0,1\}$，输出密文 c_f。*

函数 f 可表示成有限域 $GF(2)$ 上的算术电路形式。全同态加密算法是安全的，指满足语义安全。

全同态加密分类

根据计算电路的深度，全同态加密算法可以分为两种形式：

- 一是“纯”的全同态加密算法，即可以执行任意深度的电路计算算法，但是目前“纯”的全同态加密算法只能依靠同态解密技术（Bootstrap）和循环安全假设来实现；
- 二是层次型全同态加密算法，即算法只能执行深度为 L 的计算电路，算法的参数依赖于 L。

L-同态：对于一个同态加密算法 HE，在存在 $L = L(\lambda)$，如果对于深度为 L 的任何有限域 $GF(2)$ 上的算术电路 f，以及任意输入 $m_1, m_2, \cdots, m_t$ 满足以下条件：

$$\Pr[\text{HE.Dec}_{sk}(\text{HE.Eval}_{evk}(f, c_1, c_2, \cdots, c_t)) \neq f(m_1, m_2, \cdots, m_t)] = \text{negl}(\lambda),$$

其中，Dec 是 HE 的解密算法，Eval 是密文计算算法，sk 是密钥，evk 是密文计算公钥。通常称为同态加密算法 HE 是 **L-同态**。

部分同态加密（Somewhat Homomorphic Encryption, SWHE）：算法只能执行有限电路深度的密文计算，部分同态加密算法是 L 同态的。

紧凑性：如果一个同态加密算法的解密电路是独立于计算深度的，即密文的长度与计算电路的深度无关，则称该同态加密算法是紧凑的。

全同态：如果一个紧凑的同态加密算法是 L 同态的且 $L = \infty$，即可以任意多项式深度的计算电路，则称该算法是全同态的。

层次型全同态：如果某全同态加密算法的公钥/密钥生成算法中，将作为输入参数的 L 明确嵌入了电路计算深度 L，则称该算法是层次型全同态算法。

2 关键技术

格上加密算法生成的密文是一种**带噪声的密文**，即在明文里叠加噪声，因此密文计算时会导致密文中的噪声增长，当噪声增长超过某个界限时，就不能正确解密。因此，实现金全同态加密的关键是对密文计算过程中的噪声增长进行管理。目前有 3 种噪声管理技术：

- 同态解密/自举（Bootstrapping）；
- 模交换（Modulus Switching），在3.2.5阐述；
- 位展开（Bit Decomposition）。

2.1 Bootstrapping 自举

自举技术是 Gentry 实现全同态加密的奠基石。Gentry 在实现全同态加密时，注意到如果同时执行自己的解密算法，即输入的密文是对原密文中每一位加密的结果，输入的密钥是对原密文密钥每一位加密的结果，那么经过同态执行解密算法就会得到一个新的密文，这个新的密文噪声是固定的。因此，可以通过同态解密技术约束（管理）密文的噪声。

1. **密钥生成：**私钥和公钥对分别为 $(sk_1, pk_1), (sk_2, pk_2)$。
2. **加密：**使用 pk_1 对明文 m 加密 $\langle m \rangle = \text{Enc}(pk_1, m)$。注意，私钥 sk_1 能脱衣服 1⟨ ⟩
   Dec算法：输入 sk_1 和 $\langle m \rangle$，进行 k_1 次加法和 k_2 次乘法的组合，则脱掉衣服 1⟨ ⟩。
3. **加密：**使用 pk_2 对密文 $\langle m \rangle$ 加密 $[\![\langle m \rangle]\!] = \text{Enc}(pk_2, \langle m \rangle)$。注意，私钥 sk_2 能脱衣服 2⟦ ⟧，需要 k_1 次加法和 k_2 次乘法的组合。
4. **加密：**使用 pk_2 对密文 sk_1 加密 $[\![sk_1]\!] = \text{Enc}(pk_2, sk_1)$。
5. **同态运算：**密文进行同态加法和同态乘法，等价于明文做对应的加法和乘法。因此，输入 $[\![sk_1]\!]$ 和 $[\![\langle m \rangle]\!]$，进行 k_1 次同态加法和 k_2 次同态乘法的组合，则能够脱掉衣服 1⟨ ⟩，获得 $[\![m]\!]$。

等价于**同态解密：** $[\![m]\!] = \text{Dec}([\![sk_1]\!], [\![\langle m \rangle]\!])$ 用 $[\![sk_1]\!]$ 脱掉衣服 1$\langle\,\rangle$，还剩衣服 2$[\![\,]\!]$。

核心思想：步骤 3 到步骤 5，称为换衣服，但不走光（明文 m 和私钥 sk_1 不泄露）

- 假设衣服 1 密文 $\langle m_0 \rangle$，经过 n 次同态运算变为衣服 1 密文 $\langle m_n \rangle'$。此时噪声积累很严重（多了 $'$），后续仅能进行 k_1 次同态加法和 k_2 次同态乘法运算了。
- 使用步骤 3,4,5【换衣服】，执行 k_1 次同态加法和 k_2 次同态乘法，将衣服 1 密文 $\langle m_n \rangle'$ 变为衣服 2 密文 $[\![m_n]\!]$，则能够进行 n 次同态运算，变为 $[\![m_{2n}]\!]'$。此时噪声积累很严重（多了 $'$），仅剩余 k_1 次同态加法和 k_2 次同态乘法运算了。
- 再使用步骤 3,4,5，继续换衣服，则又可以同态运算 n 次，而不泄露明文 m 和私钥 sk_1。

Bootstrapping 自举：每同态运算 n 次后，则换衣服，则再同态运算 n 次，则再换衣服，以此类推。
由于每层电路（k_1 次加法和 k_2 次乘法运算的组合）都需要密钥对 (pk_1, sk_1)，所以层次型全同态加密算法的公钥 $(pk_1, pk_2, \cdots, pk_{L+1})$ 和 $(sk_1, sk_2, \cdots, sk_L)$ 构成。如果是循环安全的，每层电路就可以使用相同的密钥对，减少密钥数量。尽管同态解密是实现全同态加密的基石，但同态解密的效率很低，其复杂度为 $\mathcal{O}(\lambda^3)$。
Bootstrapping【自举/同态解密/换衣服】是一种通用的管理噪声的技术，**可以应用于所有 FHE 算法**，只要解密电路的深度小于算法本身的同态计算电路的深度。

2.2 位展开

当噪声主要依赖于某一个向量 $\mathbf{x} \in \mathbb{Z}_q^n$ 时，为降低噪声的影响，可将该向量**位展开**，即将向量中的每个元素展开成二进制形式，从而向量 $\mathbf{x}$ 的范数 $l_1(\mathbf{x})$ 的最大值从 nq 变成 $n \log q$，降低噪声。
BitDecomp(x): $\mathbf{x} \in \mathbb{Z}^n$，令 $\mathbf{w}_i \in \{0,1\}^n$，满足：$\mathbf{x} = \sum_{i=0}^{\lceil \log q \rceil - 1} 2^i \cdot \mathbf{w}_i (\text{mod } q)$，输出：

$$(\mathbf{w}_0, \mathbf{w}_1, \cdots, \mathbf{w}_{\lceil \log q \rceil - 1}) \in \{0,1\}^{n \lceil \log q \rceil}$$

BitDecomp^{-1}(y): 是 **BitDecomp**(x) 的逆运算。令：

$$\mathbf{y} = (\mathbf{w}_0, \mathbf{w}_1, \cdots, \mathbf{w}_{\lceil \log q \rceil - 1})$$

则 **BitDecomp^{-1}(y)** 输出：

$$\sum_{i=0}^{\lceil \log q \rceil - 1} 2^i \cdot \mathbf{w}_i (\mod q)$$

即使向量 $\mathbf{w}$ 中的元素不是 0 和 1，该定义依然成立。
Flatten(y): 令 **Flatten(y) = BitDecomp(BitDecomp^{-1}(y))**，则 **Flatten(y)** 是一个元素为 0 和 1 构成的向量。因此，Flatten 能够将一个元素不是 0 和 1 的向量转化成一个元素为 0 和 1 的向量，而维数不变。Gentry 等在 2013 年提出使用 Flatten 技术作为密文噪声管理。
向量**位展开**后，为了使得向量的内积保持不变，需要另外一个向量变成如下形式：
Powersof2(y): 输入 $\mathbf{y} \in \mathbb{Z}^n$，输出

$$(\mathbf{y}, 2\mathbf{y}, \cdots, 2^{\lceil \log q \rceil - 1}\mathbf{y}) \mod q \in \mathbb{Z}^{n \cdot \lceil \log q \rceil}$$

对于所有 $\mathbf{x} \in \mathbb{Z}_q^n$ 和 $\mathbf{y} \in \mathbb{Z}_q^n$，有如下性质：

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \text{BitDecomp}(\mathbf{x}), \text{Powersof2}(\mathbf{y}) \rangle = \langle \text{Powersof2}(\mathbf{x}), \text{BitDecomp}(\mathbf{y}) \rangle$$

同理，对于任何一个 $n \cdot \lceil \log q \rceil$ 维向量 $\mathbf{a}$，有：

$$\langle \mathbf{a}, \text{Powersof2}(\mathbf{y}) \rangle = \langle \text{BitDecomp}^{-1}(\mathbf{a}), \mathbf{y} \rangle = \langle \text{Flatten}(\mathbf{a}), \text{Powersof2}(\mathbf{y}) \rangle$$

3 四代全同态加密

四代 FHE 的特点与区别：

- 第一代：引入 Bootstrapping 技术，在理论上首次支持无限次同态运算。代表作为 2009 年 Gentry 和 2010 年 DGHV，均为“基于整数”分支。
- 第二代：对 arithmetic 电路，FHE 进入实用阶段。代表作为 2011 年 BGV、2012 年 LTV、2012 年 BFV（属 leveled schemes 分支）、2013 年 BLLN（属 NTRU 分支）。**再线性化**、**密钥交换**、**模交换**技术，避免使用昂贵的 Bootstrapping 技术，提升效率。
- 第三代：对 Boolean 电路，降低同态运算中的误差积累。代表作为 2013 年 GSW、2014 年 FHEW（属 fast bootstrapping 分支）。GSW 使用简单的**比特转换**技术实现 Leveled-FHE 方案。
- 第四代：支持浮点数计算。代表作为 2016 年 CKKS（属 leveled schemes 分支）、2016 年 TFHE（属 fast bootstrapping 分支）。对于浮点数、复数域上的计算，CKKS16 方案基于 RLWE 问题，所以本质上是在整数上计算。CKKS16 将浮点数、复数编码为整数，再进行后续计算。如，m 是一个浮点数，编码为整数，则乘以扩张因子 δ，再进行取整。对编码后数计算乘法，扩张因子变为 δ^2，所以需要消去一个 δ，即在 CKKS 中的 rescale 操作。

3.1 第 1 代：基于理想格和 AGDC

Gentry09 的方案涉及了较多的代数数论，方案复杂度较高。DGHV10 方案基于整数，方案简单，便于理解，但计算复杂度高，公钥尺寸非常大。

3.1.1 Gentry09

Fully homomorphic encryption using ideal lattices Fully homomorphic encryption for mathematicians

理想：$\mathbb{I}$ 是环 $\mathbb{R}$ 的一个子集，包含 0（即加法的逆元素），满足以下条件：

1. $\mathbb{I}$ 中两个元素的加法结果仍然在 $\mathbb{I}$ 中；
2. $\mathbb{I}$ 中的一个元素与 $\mathbb{R}$ 中的一个元素的乘积仍然在 $\mathbb{I}$ 中。

主理想是由一个元素生成的**理想**，即由 a 生成的主理想是 a 的倍数组成的集合。

Gentry09 方案：使用由理想 $\mathbb{I}$ 定义的整数子格 $L(\mathbb{I}) \subseteq \mathbb{Z}^m$。

- **密钥生成：**公钥矩阵 B_{pk} 和私钥矩阵 B_{sk} 分别是理想格 $L(\mathbb{I})$ 的劣质基与优质基。
- **加密：**消息是 1 个比特，$m \in \{0, 1\}$，如下计算

$$\mathbf{a} = m + 2\mathbf{e} \in \mathbb{R}^d$$

其中，**e** 是噪声，即一个随机向量，其系数属于 $\{0, \pm 1\}$，且 ± 1 取值的概率相等。密文 **c** 是点 **a** 在平行多面体 $P(B_{pk})$ 中的平移。其中，B_{pk} 是公钥，即

$$\mathbf{c} = \mathbf{a} - (\lceil \mathbf{a} B_{pk}^{-1} \rfloor B_{pk})$$

其中，$\lceil \cdot \rfloor$ 表示四舍五入到最近的整数。

分析：使用 $\lceil \mathbf{a} B_{pk}^{-1} \rfloor$ 对劣质基 B_{pk} 中的向量线性组合，获得一个随机格点；然后加上一个随机距离 **a**，获得一个不在格 L 中的随机点 **c**。

- **解密：**计算

$$\mathbf{a}' = \mathbf{c} - (\lceil \mathbf{c} B_{sk}^{-1} \rfloor B_{sk})$$

这表示将 **c** 平移到平行多面体 $\mathcal{P}(B_{sk})$。其中，B_{sk} 是私钥。然后计算 $\mathbf{a}' \mod 2$ 获得 m。

分析：使用优质基 B_{sk}，是对不在格 L 中的随机点 **c** 去掉噪声，则获得一个与随机点 **c** 最近的格点 $\lceil \mathbf{c} B_{sk}^{-1} \rfloor B_{sk}$，即解决 CVP 困难问题。然后作差，获得距离 **a**。

- **同态加法：**整数环中的加法 $\mathbf{c} + \mathbf{c}' = \mathbf{a} + \mathbf{a}' - (\lceil \mathbf{a} B_{pk}^{-1} \rfloor + \lceil \mathbf{a}' B_{pk}^{-1} \rfloor) B_{pk}$
- **解密：**基于优质基，找到最近格点，然后获得距离 $\mathbf{a} + \mathbf{a}'$。
- **同态乘法：**整数环中的乘法

$$\begin{aligned}
&\mathbf{c} \times \mathbf{c}' \\
&= \left(\mathbf{a} - (\lceil \mathbf{a} B_{pk}^{-1} \rfloor B_{pk})\right) \times \left(\mathbf{a}' - (\lceil \mathbf{a}' B_{pk}^{-1} \rfloor B_{pk})\right) \\
&= \mathbf{a}\mathbf{a}' - \mathbf{a}(\lceil \mathbf{a}' B_{pk}^{-1} \rfloor B_{pk}) - \mathbf{a}'(\lceil \mathbf{a} B_{pk}^{-1} \rfloor B_{pk}) + (\lceil \mathbf{a} B_{pk}^{-1} \rfloor B_{pk})(\lceil \mathbf{a}' B_{pk}^{-1} \rfloor B_{pk})
\end{aligned}$$

- **解密：**第 1 项是距离，后 3 项是随机格点，再累加距离 aa'，则得到随机点 cc'。所以使用优质基去掉随机点的噪声（解决 CVP 困难问题），获得最近的格点。然后再计算获得 $\mathbf{a}\mathbf{a}'$。

上述方案由于解密算法的复杂性（即无法同态评估）而无法进行引导自举（Bootstrapping 同态解密）。

为了解决该问题，Gentry 提出了一个方法，通过对原始的同态加密方案 E 进行变换，得到一个新的方案 E^*。该方案具有相同的同态能力，但具有一个简化的解密函数，从而允许 Bootstrapping。

为了降低解密算法的复杂性，Gentry 证明只需在评估密钥中添加一些关于私钥的“额外信息”。这些额外信息由一组向量 $S = \{s_i | i = 1, \ldots, S\}$ 组成，从中导出一个子集 T。私钥 sk 是 T 元素的和，评估密钥中包含的公钥信息是集合 S。该新方案 E^* 的安全性基于：集合 T 是稀疏的且是私有的，因此适用稀疏子集和困难问题（SSSP）。

该方案的安全性基于 4 个困难问题：

1. **稀疏子集和困难问题（SSSP）：**已知 n 个整数 $S = \{a_1, ..., a_n\} \subseteq \mathbb{Z}$，判断是否存在集合 S 的子集 $\mathbb{I}$，满足 $\sum_{i \in \mathbb{I}} a_i = 0$。
2. **有界距离解码困难问题（CVP 困难问题的变形）：**已知一个目标向量 **t**，使得 $\text{dist}(\mathbf{t}, L) < \alpha \lambda_1(L)$。其中，$\lambda_1(L)$ 是格 L 的第一个最短基向量的长度。求与 **t** 最接近的一个格向量 $\mathbf{v} \in L$。
3. **理想最短向量困难问题（SVP 困难问题的变形）：**在格 L 中找到一个最短的非零向量。其中，$\lambda_1(L)$ 是 $\gamma \lambda_1(L) < \lambda_2(L)$ 且 $\gamma \geq 1$。换句话说，最短向量的长度保证至少是 $\lambda_2(L)$ 的 γ 倍小。
4. **循环安全**，文中未给出证明。

安全问题：CDPR16 发现了使用主理想方案的一个漏洞：在给定一个量子多项式时间，或经典的 $2^{n^{2/3-\varepsilon}}$ 时间算法的情况下，可以进行密钥恢复攻击，攻击目标是基于主理想格的加密构造。Recovering Short Generators of Principal Ideals in Cyclotomic Rings

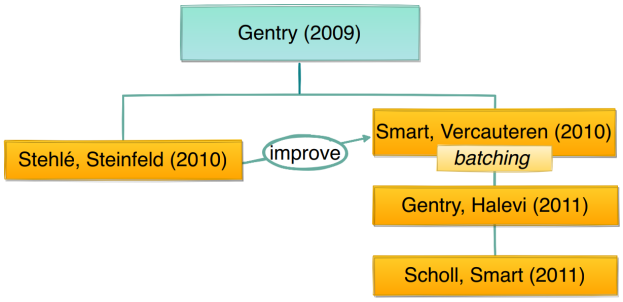

图 3: 基于理想格的方案汇总

3.1.2 DGHV10

Fully homomorphic encryption over the integers

该方案是1.2.4方案实例的一般化

- **密钥生成：**私钥为 p，即一个奇数随机整数，以及公钥 $(x_0, \ldots, x_n)$。x_0 是奇数，且 $x_0 > x_i, \forall i$

$$x_i = pq_i + r_i$$

  q_i, r_i 是随机整数。

- **加密：**消息 $m \in \mathbb{F}_2$，如下计算

$$c = (m + 2r + 2\sum_{i \in S} x_i) \mod x_0$$

  其中，r 是一个随机整数，S 是 $\{1, \ldots, n\}$ 的一个随机子集。

- **解密：**计算 $(c \mod p) \mod 2$

该方案的安全性基于稀疏子集和问题（SSSP）以及近似最大公约数（AGCD）困难问题。

- **SSSP 困难问题：**已知 n 个整数 $S = \{a_1, ..., a_n\} \subseteq \mathbb{Z}$，判断是否存在集合 S 的子集 $\mathbb{I}$，满足 $\sum_{i \in \mathbb{I}} a_i = 0$。
- **AGCD 困难问题：**已知随机选择的整数集 $\{x_0, \ldots, x_n\} \in \mathbb{Z}$，求解“共同近似除数” p。

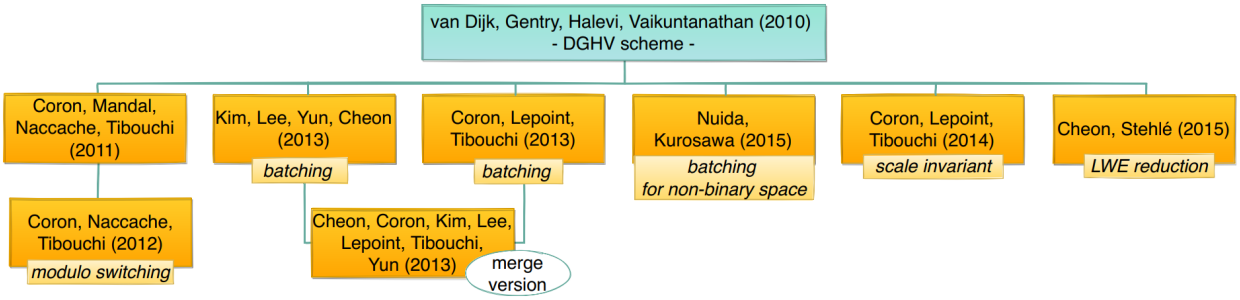

图 4: 基于 AGCD 困难问题的 FHE 方案汇总

3.2 第 2 代：基于 LWE 和 RLWE

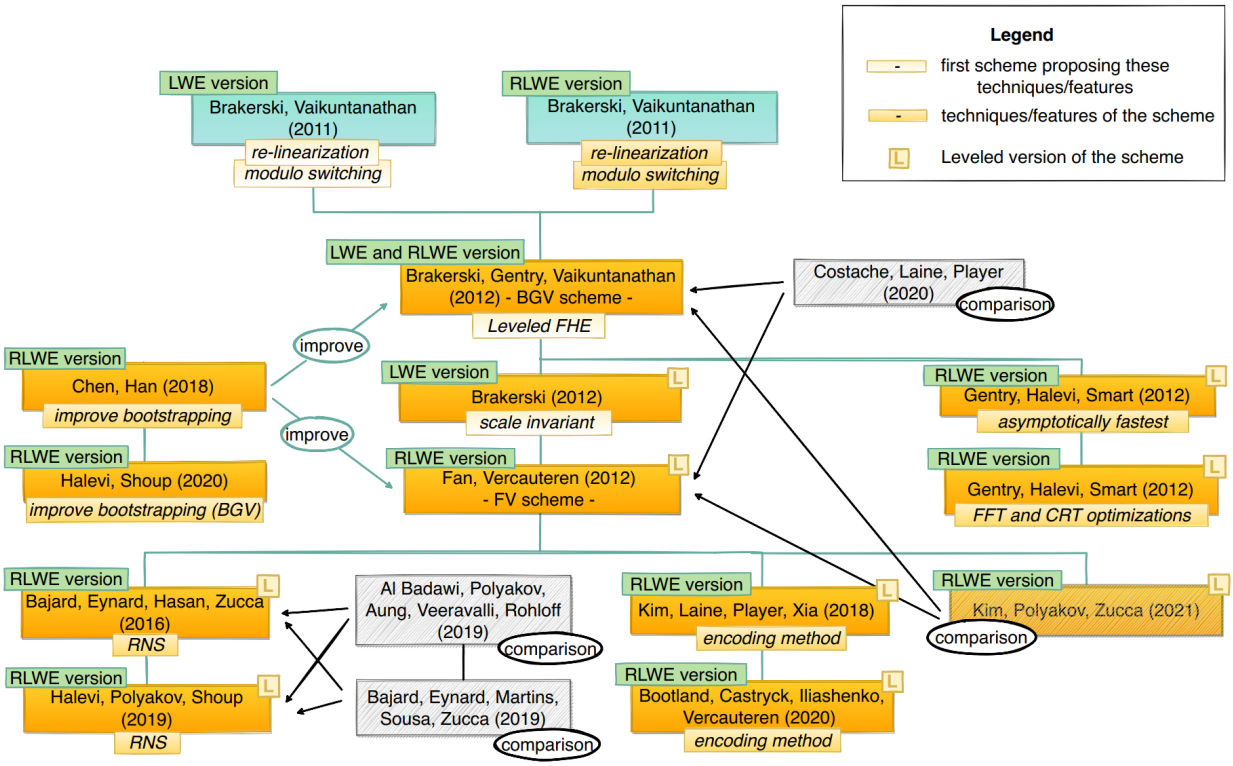

图 5: 第 2 代 FHE：基于 LWE 和 RLWE

3.2.1 预备知识：Kronecker Product

Wikipedia: Kronecker Product

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 2 \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \\ 3 \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 4 \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 5 & 0 & 10 \\ 6 & 7 & 12 & 14 \\ 0 & 15 & 0 & 20 \\ 18 & 21 & 24 & 28 \end{bmatrix}$$

- 张量积是向量积 $\mathbf{a} \otimes \mathbf{b}$；
- Kronecker 积是矩阵积 $\mathbf{A} \otimes \mathbf{B}$。

Kronecker 积有以下 9 个性质，最后 2 个是矩阵性质。

1. $A \otimes (B + C) = A \otimes B + A \otimes C$，要求 B 和 C 相同尺寸
2. $(A + B) \otimes C = A \otimes C + B \otimes C$，要求 A 和 B 相同尺寸
3. $kA \otimes B = A \otimes kB = k(A \otimes B)$
4. $(A \otimes B) \otimes C = A \otimes (B \otimes C)$
5. $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$
6. $(A \otimes B)^T = A^T \otimes B^T$

7. $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$
8. $A \otimes B \neq B \otimes A$
9. $B \otimes A = P(A \otimes B)Q$, P 和 Q 是置换矩阵
10. $(AB)^T = B^T A^T$，矩阵性质
11. $(AB)^{-1} = B^{-1}A^{-1}$，矩阵性质

3.2.2 BV 对称加密

Efficient Fully Homomorphic Encryption from (Standard) LWE BGV14 方案中的基于 LWE 的公钥加密类似。

- **对称密钥：** 选择随机数 $\mathbf{s} \in \mathbb{Z}_q^n$，则私钥为 $\bar{\mathbf{s}} = (-\mathbf{s}, 1) \in \mathbb{Z}_q^{n+1}$。
- **加密：** 消息为 $m =\in \mathbb{F}_2^k$，令 $\mathbf{m} = (m, 0..., 0)$，选择随机矩阵 $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$ 和噪声 $\mathbf{e} \in \mathbb{Z}_q^k$，$e$ 从误差分布 χ 中随机选择的，计算

$$\mathbf{c} = (\mathbf{A}, b = \langle \mathbf{A}, \mathbf{s} \rangle + 2\mathbf{e} + \mathbf{m}) \in \mathbb{Z}_q^{k \times n} \times \mathbb{Z}_q^k$$

  则密文为 $\mathbf{c} = (\mathbf{A}, \mathbf{b})$。
- **解密：** 输入私钥 $\bar{\mathbf{s}} = (-\mathbf{s}, 1)$ 和密文 $\mathbf{c} = (\mathbf{A}, \mathbf{b})$，如下计算

$$\begin{aligned}
&\langle \mathbf{c}, \bar{\mathbf{s}} \rangle \mod q \mod 2 \\
&= (\mathbf{b} - \langle \mathbf{A}, \mathbf{s} \rangle \mod q) \mod 2 \\
&= ((\langle \mathbf{A}, \mathbf{s} \rangle + 2\mathbf{e} + \mathbf{m}) - \langle \mathbf{A}, \mathbf{s} \rangle \mod q) \mod 2
\end{aligned}$$

- **同态加法：**

$$\begin{aligned}
&\mathbf{c} + \mathbf{c}' = (\mathbf{A}, \mathbf{b}) + (\mathbf{A}', \mathbf{b}') \\
&= (\mathbf{A} + \mathbf{A}', \langle \mathbf{A}, \mathbf{s} \rangle + 2\mathbf{e} + \mathbf{m} + \langle \mathbf{A}', \mathbf{s} \rangle + 2\mathbf{e}' + \mathbf{m}') \\
&= (\mathbf{A} + \mathbf{A}', \langle \mathbf{A} + \mathbf{A}', \mathbf{s} \rangle + 2(\mathbf{e} + \mathbf{e}') + \mathbf{m} + \mathbf{m}') \\
&= (\mathbf{A}'', \langle \mathbf{A}'', \mathbf{s} \rangle + 2\mathbf{e}'' + \mathbf{m}'') \\
&= (\mathbf{A}'', \mathbf{b}'') \\
&= \mathbf{c}''
\end{aligned}$$

  其中，$\mathbf{m}'' = \mathbf{m} + \mathbf{m}'$。注意，$\mathbf{e}'' = \mathbf{e} + \mathbf{e}'$ 噪声有积累。同态加法情况下影响较小。
- **解密：** 输入私钥 $\bar{\mathbf{s}} = (-\mathbf{s}, 1)$ 和密文 $\mathbf{c}'' = (\mathbf{A}'', \mathbf{b}'')$，如下计算

$$\mathbf{m} + \mathbf{m}' := \langle \mathbf{c}'', \bar{\mathbf{s}} \rangle = (\mathbf{b}'' - \langle \mathbf{A}'', \mathbf{s} \rangle \mod q) \mod 2$$

- **同态乘法：** $\mathbf{c} \otimes \mathbf{c}'$，称为 Kronecker 积（Kronecker Product），是外积/张量积从向量到矩阵的推广

- **解密：** 私钥 $\bar{\mathbf{s}}$ 扩展为新私钥 $\bar{\mathbf{s}} \otimes \bar{\mathbf{s}}$。注意：私钥长度增加了。如下计算

$$\begin{aligned}
&\langle \mathbf{c} \otimes \mathbf{c}', \bar{\mathbf{s}} \otimes \bar{\mathbf{s}} \rangle \\
&= (\mathbf{c} \otimes \mathbf{c}')^T (\bar{\mathbf{s}} \otimes \bar{\mathbf{s}}) \\
&= (\mathbf{c}^T \otimes \mathbf{c}'^T)(\bar{\mathbf{s}} \otimes \bar{\mathbf{s}}) \\
&= (\mathbf{c}^T \bar{\mathbf{s}}) \otimes (\mathbf{c}'^T \bar{\mathbf{s}}) \\
&= \langle \mathbf{c}, \bar{\mathbf{s}} \rangle \otimes \langle \mathbf{c}', \bar{\mathbf{s}} \rangle \\
&= \langle \mathbf{c}, \bar{\mathbf{s}} \rangle \cdot \langle \mathbf{c}', \bar{\mathbf{s}} \rangle \\
&= \mathbf{m} \cdot \mathbf{m}'
\end{aligned}$$

其中，等号 1：内积；等号 2：性质 6；等号 3：性质 5；等号 4：内积；等号 5：Kronecker 内积退化；等号 6：常规解密。

论文引入了 2 种新技术：**再线性化**和**维度模缩小**。

- **再线性化**是必要的，以将乘法密文大小从接近 $n^2/2$ 降到常规大小，即 $n+1$。为了实现这一减少，通过“加密” 对称密钥的所有项来转换 $c_1 \cdot c_2$ 的二次方程为线性方程。随后，Brakerski、Gentry 和 Vaikuntanathan (Leveled) Fully Homomorphic Encryption without Bootstrapping 将这一技术称为**密钥交换**（key switching）。在3.2.4节阐述。
- **维度模缩小**，即**模交换技术**，将同态加密方案（SHE）转换为完全同态加密方案，通过将密文 c 对模 q 的取模转换为密文 c' 对模 p 的取模。其中，p 比 q 小得多。具体而言，先将 $\mathbb{Z}_q$ 中的每个元素乘以 p/q，然后取最近的整数。此操作的一个有趣副作用是密文中的误差减少。由于采用了模切换，噪声增长较低，这使得可以同态评估解密电路，而无需 Gentry 提出的压缩方法，使得 SSSP 假设不再是必需的。在3.2.5节阐述。

3.2.3 BGV14

(Leveled) fully homomorphic encryption without bootstrapping

设 d 为 2 的幂，q 为奇正整数模数，χ 为 R 上的误差分布。其中，$R = \mathbb{Z}[x]/\langle x^d+1 \rangle$。设 B 为 χ 输出元素长度的界限（以极高的概率），并且 B 被设置为尽可能小以保持安全性。对于任意自然数 p，$R_p = \mathbb{Z}_p[x]/\langle x^d+1 \rangle$。该方案的工作流程如下：

- **密钥生成：** 输入安全参数 λ，随机选择一个小的秘密元素 $s \in \chi$，并设置私钥 $sk = \mathbf{s} = (1, s) \in R_q^2$。随机均匀生成一个 $a' \in R_q$，然后计算 $b = a's + 2e$。其中，e 是从 χ 中随机选择的误差。输出 sk 和公钥 $pk = \mathbf{a} = (b, -a')$。以下等式成立

$$\langle \mathbf{a}, \mathbf{s} \rangle = \langle (b, -a'), (1, s) \rangle = \langle (a's + 2e, -a'), (1, s) \rangle = 2e$$

- **加密：** 输入公钥 $pk \in R_q^2$ 和消息 $m \in R_2$，将 m 转换为向量 $\mathbf{m} = (m, 0) \in R_q^2$，并选择噪声 $r, e_0, e_1 \in \chi$。输出密文为

$$\begin{aligned}
\mathbf{c} &= \mathbf{m} + 2(e_0, e_1) + \mathbf{a}r \\
&= (m + 2e_0 + br, 2e_1 - a'r) \in R_q^2 \\
&= (c_0, c_1)
\end{aligned}$$

- **解密：**输入私钥 $s \in R_q^2$ 和密文 $\mathbf{c} \in R_q^2$，计算内积

$$\begin{aligned}\langle \mathbf{c}, \mathbf{s} \rangle &= c_0 + c_1 s \\ &= m + 2e_0 + br + (2e_1 - a'r)s \\ &= m + 2e_0 + (a's + 2e)r + (2e_1 - a'r)s \\ &= m + 2e_0 + 2e_1 s + 2er\end{aligned}$$

  并输出 $((m + 2(e_0 + e_1 s + er)) \mod q) \mod 2 = m$。

  注意：解密能正常工作是因为 e, e_0, e_1 和 s 足够小（因为它们是 χ 的元素）。

- **同态加法：**对应相加

$$\begin{aligned}\mathbf{c} + \mathbf{c}' &= (c_0, c_1) + (c_0', c_1') \\ &= (m + 2e_0 + br, 2e_1 - a'r) + (m' + 2e_0' + br', 2e_1' - a'r') \\ &= (m + m' + 2(e_0 + e_0') + b(r + r'), 2(e_1 + e_1') - a'(r + r')) \\ &= (m'' + 2e_0'' + br'', 2e_1'' - a'r'') \\ &= (c_0'', c_1'') \\ &= \mathbf{c}''\end{aligned}$$

- **解密：**

$$\begin{aligned}\langle \mathbf{c}'', \mathbf{s} \rangle &= c_0'' + c_1'' s \\ &= m'' + 2e_0'' + br'' + (2e_1'' - a'r'')s \\ &= m'' + 2e_0'' + (a's + 2e)r'' + (2e_1'' - a'r'')s \\ &= m'' + 2e_0'' + 2e_1'' s + 2er''\end{aligned}$$

  再 $\mod q \mod 2$，则获得 $m'' = m + m'$。

  因此，只要结果中的误差不与模数 q 重叠，解密就能正常工作。

- **同态乘法：** $\mathbf{c} \otimes \mathbf{c}'$

- **解密：**私钥 $\mathbf{s}$ 扩展为新私钥 $\mathbf{s} \otimes \mathbf{s}$。注意：私钥长度增加了。如下解密：

$$\begin{aligned}&\langle \mathbf{c} \otimes \mathbf{c}', \mathbf{s} \otimes \mathbf{s} \rangle \\ &= (\mathbf{c} \otimes \mathbf{c}')^T (\mathbf{s} \otimes \mathbf{s}) \\ &= (\mathbf{c}^T \otimes \mathbf{c}'^T)(\mathbf{s} \otimes \mathbf{s}) \\ &= (\mathbf{c}^T \mathbf{s}) \otimes (\mathbf{c}'^T \mathbf{s}) \\ &= \langle \mathbf{c}, \mathbf{s} \rangle \otimes \langle \mathbf{c}', \mathbf{s} \rangle \\ &= \langle \mathbf{c}, \mathbf{s} \rangle \cdot \langle \mathbf{c}', \mathbf{s} \rangle \\ &= m \cdot m'\end{aligned}$$

3.2.4 核心技术：密钥交换

解密分析：

$$\begin{aligned}
&\langle \mathbf{c} \otimes \mathbf{c}', \mathbf{s} \otimes \mathbf{s} \rangle \\
&= \langle \mathbf{c}, \mathbf{s} \rangle \cdot \langle \mathbf{c}', \mathbf{s} \rangle \\
&= (c_0 + c_1 s)(c'_0 + c'_1 s) \\
&= c_0 c'_0 + (c_0 c'_1 + c_1 c'_0)s + c_1 c'_1 s^2 \\
&= d_0 + d_1 s + d_2 s^2 \\
&= \langle (d_0, d_1, d_2), (1, s, s^2) \rangle
\end{aligned}$$

解密过程可以看系数为 (d_0, d_1, d_2) 的多项式，在横坐标 s 处的求值。

存在问题： 私钥 $\mathbf{s}$ 扩展为新私钥 $\mathbf{s} \otimes \mathbf{s}$，同态乘法一次，则需要私钥扩展一次。

解决方案： 将 s^2 加密到 s 下，可实现密文中的 $d_2 s^2$ 转换为 $\bar{c}_0 + \bar{c}_1 s$。具体而言，令消息 $m = s^2$，则 $\mathbf{m} = (s^2, 0)$。

选择随机数 $r, e_0, e_1 \in \chi$，如下加密：

$$\mathbf{c} = (s^2, 0) + 2(e_0, e_1) + (b, -a')r = (s^2 + 2e_0 + br, -(a'r - 2e_1)) \approx (s^2 + \alpha s, -\alpha) = (\beta, -\alpha).$$

其中，$\alpha = a'r - 2e_1$，

$s^2 + 2e_0 + br = s^2 + 2e_0 + (a's + 2e)r = s^2 + 2e_0 + a'sr + 2er \approx s^2 + a'rs - 2e_1 s = s^2 + \alpha s$，仅去掉了 $2e_0$。

因此，$s^2 = \beta - \alpha s$，使得扩展密文 (d_0, d_1, d_2) 变成一个正常密文 $(\bar{c}_0, \bar{c}_1)$。

$$d_0 + d_1 s + d_2 s^2 = d_0 + d_1 s + d_2(\beta - \alpha s) = \bar{c}_0 + \bar{c}_1 s$$

- **扩展密文** $\mathbf{c} \otimes \mathbf{c}' = (c_0, c_1) \otimes (c'_0, c'_1) = (d_0, d_1, d_2)$，对应的扩展私钥为 $\mathbf{s} \otimes \mathbf{s} = (1, s, s^2)$。
- **正常密文** $(\bar{c}_0, \bar{c}_1)$，对应的正常密钥为 $\mathbf{s} = (1, s)$。

因此，**密钥交换**技术把**扩展密文**变成了**正常密文**，使得**解密密钥不变**。

3.2.5 核心技术：模交换

同态乘法：

$$\begin{aligned}
\mathbf{c} \otimes \mathbf{c}' &= (c_0, c_1) \otimes (c'_0, c'_1) = (c_0 c'_0, c_0 c'_1, c_1 c'_0, c_1 c'_1) \\
&= (m + 2e_0 + br)(m' + 2e'_0 + br') + (m + 2e_0 + br)(2e'_1 - a'r') \\
&\quad + (m' + 2e'_0 + br')(2e_1 - a'r) + (2e_1 - ar)(2e'_1 - a'r')
\end{aligned}$$

噪声为 $(r, e_0, e_1), (r', e'_0, e'_1) \in \chi$

存在问题： 有两个噪声值为 $||\chi||$ 的密文，相乘后噪声近似为 $||\chi||^2$，则经过 $\log k$ 层同态乘法，噪声积累就达到上限 $||\chi||^k \mod q$，则不能继续运算了。

解决方案： 每次同态乘法运算完后对噪声乘以 $1/||\chi||$，则噪声减低为 $||\chi||$。此时模缩减为 $q/||\chi||$。因此，模每缩减一次，则可以进行 $\log k$ 层同态乘法运算。相比原来，仅能进行 $\log k$ 层同态乘法，是指数级提高。

3.2.6 FV12

Somewhat Practical Fully Homomorphic Encryption

令 $R = \mathbb{Z}[x]/\langle x^d + 1\rangle$。其中，$d$ 是 2 的幂。令 q 和 p 为正整数，令放大因子 $\Delta = \lfloor q/p \rfloor$ 和 $r_t(q) = q \mod p$。对于任何自然数 t，令 $R_t = \mathbb{Z}_t[x]/\langle x^d + 1\rangle$。令 χ 是 R_q 上的 B 有界概率分布。FV 方案的如下：

- **密钥生成：** 输入安全参数 λ，输出小的随机数作为私钥 $sk = s \in \chi$。选择随机数 $a \in R_q$，并计算 $-(a \cdot s + e) \mod q$。其中，$e \in \chi$ 是噪声。输出 $sk = s \in \chi$ 和 $pk = (p_0, p_1) = (-(a \cdot s + e) \mod q, a)$。
- **加密：** 输入消息 $m \in R_p$ 和公钥 $pk \in R_q^2$。选择小的随机数 $u, e_1, e_2 \in \chi$，如下计算：

$$c_0 = (p_0 \cdot u + e_1 + \Delta \cdot m) \mod q,$$
$$c_1 = (p_1 \cdot u + e_2) \mod q$$

  输出密文 $c = (c_0, c_1)$。
- **解密：** 输入秘密密钥 $s \in R_q$ 和密文 $c \in R_q^2$，如下解密

$$m := c(s) = \lceil \frac{p \cdot (c_0 + c_1 \cdot s) \mod q}{q} \rfloor \mod p$$

  公式展开：

$$\begin{aligned}
&\frac{p[c_0 + c_1 \cdot s] \mod q}{q} \\
&= \frac{p[p_0 \cdot u + e_1 + \Delta \cdot m + p_1 \cdot u \cdot s + e_2 \cdot s] \mod q}{q} \\
&= \frac{p[-(a \cdot s + e\cdot)u + e_1 + \Delta \cdot m + a \cdot u \cdot s + e_2 \cdot s] \mod q}{q} \\
&= \frac{p[\Delta \cdot m + e \cdot u + e_1 + e_2 \cdot s] \mod q}{q} \\
&= \frac{[q \cdot m + pe \cdot u + pe_1 + pe_2 \cdot s] \mod q}{q} \\
&= m + \frac{p[e \cdot u + e_1 + e_2 \cdot s] \mod q}{q}
\end{aligned}$$

  密文 c 被视为在 s 处评估的多项式，即 $c(s)$，而不是一个具有两个分量的向量。
- **同态加法：** $c + c' = (c_0 + c_0', c_1 + c_1') = (c_0'', c_1'')$
- **解密：**

$$\begin{aligned}
&\frac{p[c_0'' + c_1'' \cdot s] \mod q}{q} \\
&= \frac{p[(c_0 + c_0') + (c_1 + c_1') \cdot s] \mod q}{q} \\
&= \frac{p[(p_0 \cdot u + e_1 + \Delta \cdot m + p_1 \cdot u \cdot s + e_2 \cdot s) + (p_0 \cdot u' + e_1' + \Delta \cdot m' + p_1 \cdot u' \cdot s + e_2' \cdot s)] \mod q}{q} \\
&= \frac{p[-(a \cdot s + e\cdot)(u + u') + (e_1 + e_1') + \Delta \cdot (m + m') + a \cdot (u + u') \cdot s + (e_2 + e_2') \cdot s] \mod q}{q} \\
&= \frac{p[\Delta \cdot (m + m') + e \cdot (u + u') + (e_1 + e_1') + (e_2 + e_2') \cdot s] \mod q}{q} \\
&= \frac{[q \cdot (m + m') + pe \cdot (u + u') + p(e_1 + e_1') + p(e_2 + e_2') \cdot s] \mod q}{q} \\
&= (m + m') + \frac{p[e \cdot (u + u') + (e_1 + e_1') + (e_2 + e_2') \cdot s] \mod q}{q}
\end{aligned}$$

- **同态乘法：** $c \otimes c' = (c_0 c_0', c_0 c_1', c_1 c_0', c_1 c_1')$

- **解密：**两个密文的乘积给出的结果是一个二次多项式：

$$\begin{aligned} & c(s) \cdot c'(s) \\ & = (c_0 + c_1 \cdot s) \cdot (c'_0 + c'_1 \cdot s) \\ & = c_0 c'_0 + (c_0 c'_1 + c_1 c'_0) s + c_1 c'_1 \cdot s^2 \\ & = \alpha_0 + \alpha_1 \cdot s + \alpha_2 \cdot s^2 \\ & = \beta_0 + \beta_1 \cdot s \end{aligned}$$

最后一步是**密钥交换**。

3.2.7 LTV13

On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption

- **密钥生成：**选择两个小的随机多项式 $f', g \in \chi$。其中，χ 是 $R = \mathbb{Z}[x]/\langle x^d + 1\rangle$ 上的 B 有界分布，且 d 是 2 的幂。私钥为 $f = 2f' + 1 \in R$。其中，$f \equiv 1 \mod 2$，且 f 在 R_q 中是可逆的。公钥为 $h = 2gf^{-1} \mod q \in R_q$。
- **加密：**输入消息为 $m \in \mathbb{F}_2$，随机小元素 $r, e \in \chi$，如下计算

$$c = m + hr + 2e \in R_q$$

则密文 c 是卷积多项式环 $R_q = \mathbb{Z}_q[x]/\langle x^d + 1\rangle$ 中的一个元素。

- **解密：**输入私钥 f 和密文 c，如下解密

$$\begin{aligned} & (fc \mod q) \mod 2 \\ & = f(m + hr + 2e) \mod q \mod 2 \\ & = fm + fhr + 2fe \mod q \mod 2 \\ & = fm + 2(gr + fe) \mod q \mod 2 \\ & = fm + 2v \mod q \mod 2 \\ & = (2f' + 1)m + 2v \mod 2 \\ & = m \end{aligned}$$

存在一个关键等式：

$$fc = mf + 2v \in R_q$$

3.2.8 核心技术：提升维数

上述环 LEW 加密方案中，私钥的维数为 k，令 $\ell = \lceil \log q \rceil$。其中，$q$ 是加密方案的模，则 powerof2(sk) 的维数为 $k\ell$。

- **加密：** 消息为 $m \in \mathbb{F}_2$，如下加密：

$$\begin{aligned}
&\mathbf{c}_{1,1} := \mathrm{Enc_{pk}}(\mathrm{m}, 0, ..., 0), ..., \mathbf{c}_{1,\ell} := \mathrm{Enc_{pk}}(\mathrm{m} \cdot 2^{\ell-1}, 0, ..., 0) \\
&\mathbf{c}_{2,1} := \mathrm{Enc_{pk}}(0, \mathrm{m}, 0, ..., 0), ..., \mathbf{c}_{2,\ell} := \mathrm{Enc_{pk}}(0, \mathrm{m} \cdot 2^{\ell-1}, 0, ..., 0) \\
&... \\
&\mathbf{c}_{k,1} := \mathrm{Enc_{pk}}(0, ..., 0, \mathrm{m}), ..., \mathbf{c}_{\mathrm{k},\ell} := \mathrm{Enc_{pk}}(0, ..., 0, \mathrm{m} \cdot 2^{\ell-1})
\end{aligned}$$

  则密文是 $k\ell \times k$ 的矩阵。
- **解密：** 使用私钥对密文矩阵中的某个密文解密即可。

3.2.9 LTV13：维数提升 + 位展开

- **加密：** 输入消息为 $m \in \mathbb{F}_2$，随机小元素 $r_i, e_i \in \chi$，如下计算

$$c_i = m \cdot 2^i + hr_i + 2e_i \in R_q$$

  则密文 $\mathbf{c} = (c_1, ..., c_\ell) \in R_q^\ell$。
- **解密：** 输入私钥 f 和密文中的第一项 c_1，如下解密

$$\begin{aligned}
&(fc_1 \mod q) \mod 2 \\
&= f(m + hr_1 + 2e_1) \mod q \mod 2 \\
&= fm + fhr_1 + 2fe_1 \mod q \mod 2 \\
&= fm + 2(gr_1 + fe_1) \mod q \mod 2 \\
&= fm + 2v \mod q \mod 2 \\
&= (2f' + 1)m + 2v \mod 2 \\
&= m
\end{aligned}$$

  存在一个关键等式：

$$fc_1 = mf + 2v \in R_q$$

  对关键等式一般化：

$$f\mathbf{c} = (c_1 f, ..., c_\ell f)^T = (mf + 2v, ..., m \cdot 2^i f + 2v) = m \cdot \mathrm{Powersof2}(f)^T + 2v \in R_q$$

- **同态加法：** $\mathbf{c} + \mathbf{c}' \in R_q^\ell$
- **解密：** 先忽略 $\mod q \mod 2$

$$\begin{aligned}
&f(\mathbf{c} + \mathbf{c}') \\
&= f\mathbf{c} + f\mathbf{c}' \\
&= (m \cdot \mathrm{Powersof2}(f)^T + 2v) + (m' \cdot \mathrm{Powersof2}(f)^T + 2v') \\
&= (m + m') \cdot \mathrm{Powersof2}(f)^T + 2v + 2v'
\end{aligned}$$

  取第一个元素，就是明文。
- **同态乘法：** $\mathbf{BitDecomp}(\mathbf{c}) \bullet \mathbf{c} \in R_q^\ell$

- **解密：**

$$
\begin{aligned}
&(\mathbf{BitDecomp}(\mathbf{c}) \bullet \mathbf{c}')f \\
&= \mathbf{BitDecomp}(\mathbf{c}) \cdot (m' \cdot \text{Powersof2}(f)^T + 2v') \\
&= m'\mathbf{c}f + 2v'\mathbf{BitDecomp}(\mathbf{c}) \\
&= m'(m \cdot \text{Powersof2}(f)^T + 2v) + 2v'\mathbf{BitDecomp}(\mathbf{c}) \\
&= m'm \cdot \text{Powersof2}(f)^T + 2vm' + 2v'\mathbf{BitDecomp}(\mathbf{c})
\end{aligned}
$$

去掉噪声，取出明文 mm'。

相关方案发展脉络如图6所示。

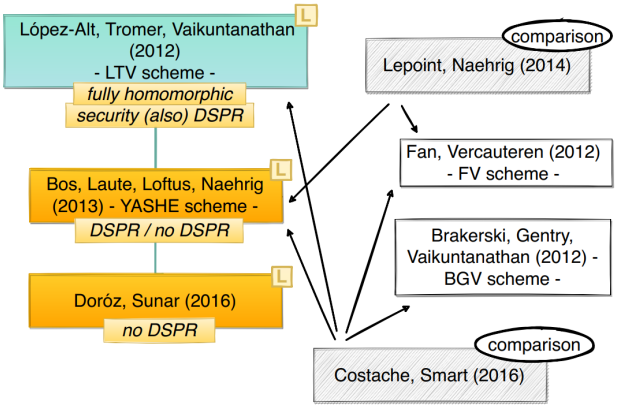

图 6: 第 2 代 FHE：基于 NTRU

3.3 第 3 代：基于 LWE 和 RLWE

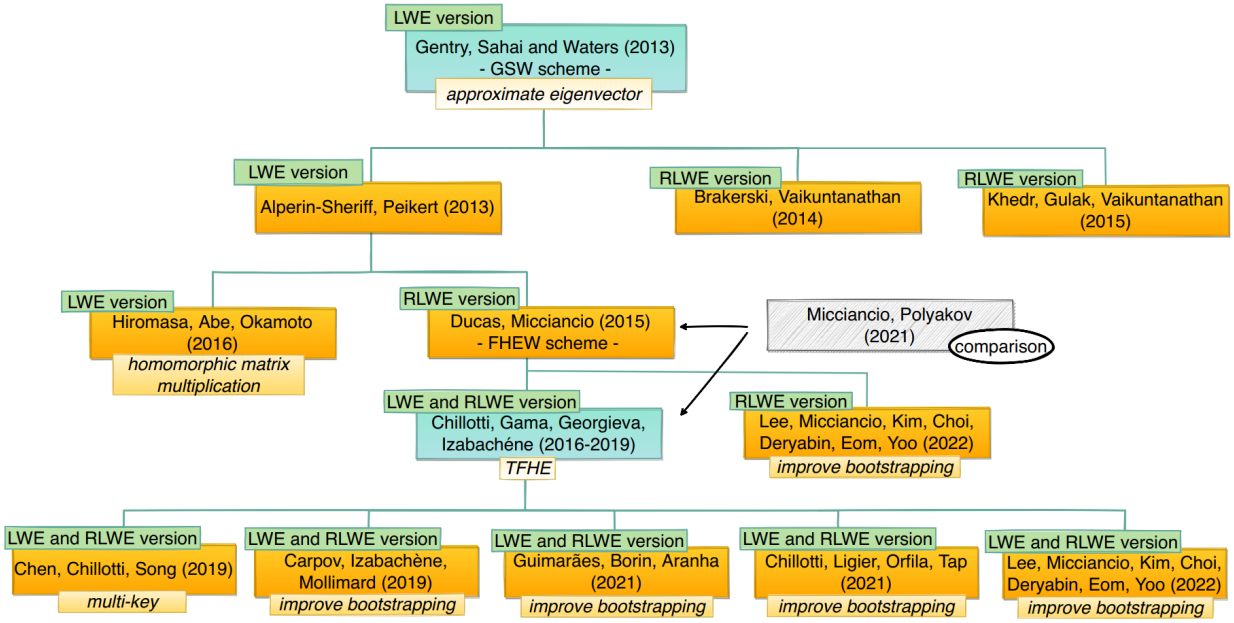

图 7: 第 3 代：基于 LWE 和 RLWE

3.3.1 GSW13

Homomorphic Encryption from Learning with Errors:Conceptually-Simpler, Asymptotically-Faster, Attribute-Based

安全参数为 λ，电路深度为 L。模为 q、噪声分布 χ、维数 n 以及 m。其中，分布 χ 是 $\mathbb{Z}$ 上的噪声高斯分布，$m = 2n\log q$。令 $l = \lceil \log q \rceil$,$N = (n+1)l$。

- **密钥生成：**选择随机向量 $\mathbf{s}' \leftarrow \mathbb{Z}_q^n$，私钥为 $sk = \mathbf{s} = (1, \mathbf{s}') \in \mathbb{Z}_q^{n+1}$。有以下等式成立：

$$\text{Powerof2}(\mathbf{s}) = (\mathbf{s}, 2\mathbf{s}, \cdots, 2^{l-1}\mathbf{s}) \mod q = (1, 2, \cdots, 2^{l-1}, \mathbf{s}', 2\mathbf{s}', \cdots, 2^{l-1}\mathbf{s}') \mod q$$

选取随机矩阵 $\mathbf{A}' \leftarrow \mathbb{Z}_q^{m\times n}$ 和噪声向量 $\mathbf{e} \leftarrow \chi^m$，计算

$$\mathbf{b} := \mathbf{A}' \cdot \mathbf{s}' + \mathbf{e}$$

令 $\mathbf{A}$ 为 $n+1$ 列矩阵，由向量 $\mathbf{b}$ 和矩阵 $\mathbf{A}'$ 构成

$$\mathbf{A} = [\mathbf{b}| - \mathbf{A}'] \in \mathbb{Z}_q^{m\times(n+1)}$$

公钥为 $pk = \mathbf{A}$。

注意，有以下等式成立

$$\mathbf{A} \cdot \mathbf{s} = [\mathbf{b}| - \mathbf{A}'] \cdot (1, \mathbf{s}') = (\mathbf{A}' \cdot \mathbf{s}' + \mathbf{e}) - \mathbf{A}'\mathbf{s}' = \mathbf{e}$$

- **加密：**消息为 $m \in \{0, 1\}$，选取随机矩阵 $\mathbf{R} \in \{0, 1\}^{N\times m}$，如下计算：

$$\mathbf{C} := m \cdot \mathbf{I}_N + \mathbf{R} \cdot \mathbf{A}$$

其中，$\mathbf{I}_N$ 是 $N \times N$ 的单位矩阵。则密文为 $\mathbf{C}$。为满足同态性，密文长度不变，通常记为 Flatten($\mathbf{C}$)。

- **解密：**输入私钥 $\mathbf{s}$ 和密文 $\mathbf{C}$，如下计算：

$$\mathbf{Cs} = (m \cdot \mathbf{I}_N + \mathbf{R} \cdot \mathbf{A})\mathbf{s} = m \cdot \mathbf{I}_N\mathbf{s} + \mathbf{Re} = m \cdot \mathbf{I}_N\mathbf{s} + \mathbf{e}^* = m \cdot \text{Powersof2}(\mathbf{s}) + \mathbf{e}^*$$

其中，$m\mathbf{I}_N\mathbf{s} = (m, ms')$ 取出 m，$\mathbf{R} \cdot \mathbf{e} = \mathbf{e}^* \approx 0$。

- **同态加法：**

$$\mathbf{C} + \mathbf{C}' = (m + m') \cdot \mathbf{I}_N + (\mathbf{R} + \mathbf{R}') \cdot \mathbf{A}$$

通常记为 Flatten($\mathbf{C} + \mathbf{C}'$)。

- **解密：**

$$(\mathbf{C} + \mathbf{C}')\mathbf{s} = ((m + m') \cdot \mathbf{I}_N + (\mathbf{R} + \mathbf{R}') \cdot \mathbf{A})\mathbf{s} = (m + m') \cdot \mathbf{I}_N\mathbf{s} + (\mathbf{R} + \mathbf{R}')\mathbf{e}$$

- **同态乘法：**

$$\mathbf{C} \bullet \mathbf{C}' = (m \cdot \mathbf{I}_N + \mathbf{R} \cdot \mathbf{A}) \bullet (m' \cdot \mathbf{I}_N + \mathbf{R}' \cdot \mathbf{A})$$

通常记为 Flatten($\mathbf{C} \bullet \mathbf{C}'$)。

- **解密：**由前面的解密步骤可知：$\mathbf{Cs} = m \cdot \text{Powersof2}(\mathbf{s}) + \mathbf{e}^*$，所以如下解密：

$$\begin{aligned}
&(\mathbf{C} \bullet \mathbf{C}')\mathbf{s} \\
&= \mathbf{C} \bullet (m' \cdot \text{Powersof2}(\mathbf{s}) + \mathbf{e}') \\
&= m'\mathbf{C}\text{Powersof2}(\mathbf{s}) + \mathbf{Ce}' \\
&= m'(m\text{Powersof2}(\mathbf{s}) + \mathbf{e}) + \mathbf{Ce}' \\
&= m'm\text{Powersof2}(\mathbf{s}) + m'\mathbf{e} + \mathbf{Ce}'
\end{aligned}$$

当 $m'\mathbf{e} + \mathbf{Ce}' \leq (N+1)E = ((n+1)l+1) \cdot 2nB \log q$ 时，可解密成功。经过深度为 L 的电路计算后，结果密文的噪声至多为 $(N+1)^L E$。因此，需满足一下条件才能正确解密：

$$(N+1)^L E = ((n+1)l+1)^L \cdot 2nB \log q < q/8$$

GSW 方案的主要缺点是较高的通信成本（密文相对于对应的明文较大）和计算复杂性。为了减少计算开销，提出了各种优化方法以改进启动过程（图7）。

3.3.2 TRLWE18

TFHE: Fast Fully Homomorphic Encryption over the Torus

令 $R = \mathbb{Z}[x]/\langle x^d + 1\rangle$。其中，$d$ 是 2 的幂次方。令 $T = \mathbb{R}[x]/\langle x^d + 1\rangle \mod 1$ 和 $R_2 = \mathbb{F}_2[x]/\langle x^d + 1\rangle$，即 R_2 中的任何元素都是具有二进制系数的 R 中的多项式。

TRLWE18 方案构造如下：

- **密钥生成：**输入安全参数 λ，输出小的私钥 $\mathbf{s} \in R_2^n$。
- **加密：**输入私钥 $\mathbf{s} \in R_2^n$，错误参数 α 和消息 $m \in T$。然后，选择一个均匀随机的掩码 $a \in T^n$，并选择一个小的噪声 $e \in \chi$。其中，χ 是一个 B 有界分布，则密文为：

$$\mathbf{c} := (\mathbf{a}, b) = (\mathbf{a}, \mathbf{s} \cdot \mathbf{a} + m + e) \in T^n \times T$$

- **解密：**输入私钥 $s \in R_2^n$ 和密文 $\mathbf{c} \in T^{n+1}$, 计算密文 $\mathbf{c}$ 的秘密线性 κ-Lipschitz 函数 φ_s（称为相位）。该相位 $\varphi_s : T^n \times T \to T$ 满足：

$$\varphi_s(\mathbf{a}, b) = b - s \cdot \mathbf{a}$$

  注意，该函数由私钥 $\mathbf{s} \in R_2^n$ 参数化。相位 $\varphi_{\mathbf{s}}(\mathbf{c})$ 接近实际的消息：

$$\varphi_{\mathbf{s}}(\mathbf{c}) = \mathbf{s} \cdot \mathbf{a} + m + e - \mathbf{s} \cdot \mathbf{a} = m + e$$

  最后，将 $\varphi_s(c)$ 四舍五入到消息空间 $M \subset T$ 中的最近点。
- **（同态）密文的线性组合：**设 $\mathbf{c}_1, \ldots, \mathbf{c}_p$ 为 p 个独立的密文，具有相同的密钥 $\mathbf{s}$，并且设 $f_1, \ldots, f_p$ 为 R 中的整数多项式。如下构造线性组合的密文：

$$\mathbf{c} = \sum_{i=1}^{p} f_i \cdot \mathbf{c}_i$$

  要求误差幅度保持小于 $1/4$，$||e||_\infty \leq 1/4$。

- **解密：**

$$\mathrm{Dec}_{\mathbf{s}}(\mathbf{c}) = \sum_{i=1}^{p} f_i \cdot \mathrm{Dec}_{\mathbf{s}}(\mathbf{c}_i)$$

密文可以线性组合，从而得到一个新的密文，解密后可以获得明文的线性组合。

3.4 第 4 代：基于 LWE 和 RLWE

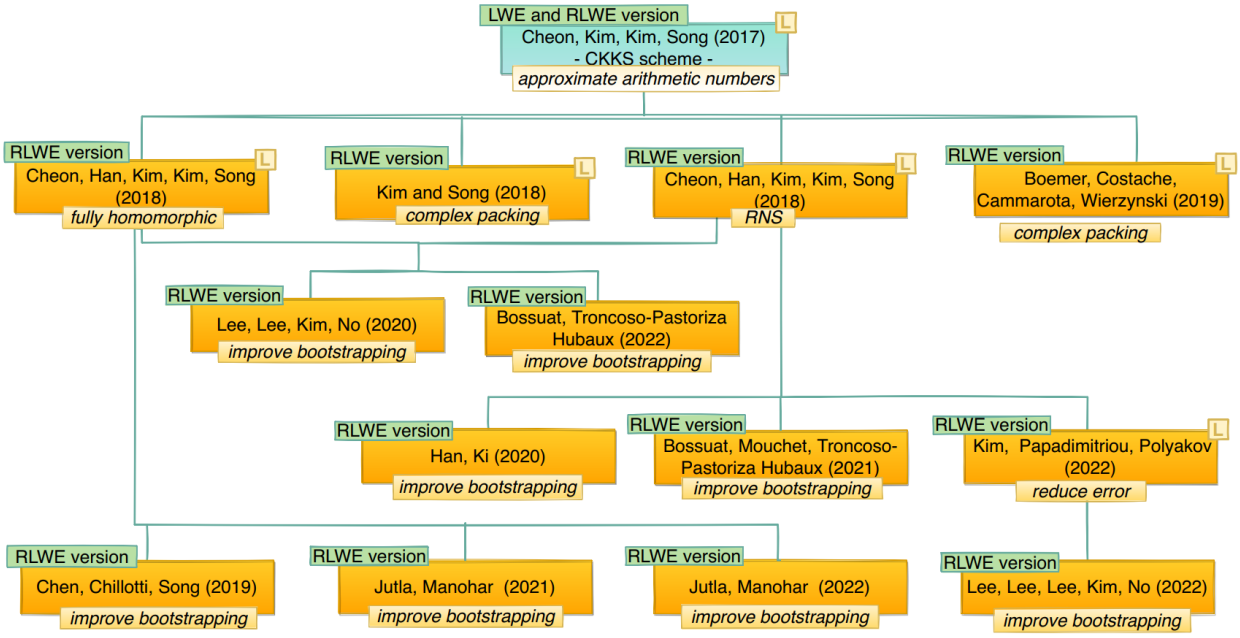

图 8: 基于基于 LWE 和 RLWE 的第四代 FHE

3.4.1 CKKS16

Homomorphic Encryption for Arithmetic of Approximate Numbers

令 $R = \mathbb{Z}[x]/\langle x^d + 1\rangle$ 且 $d = 2^M$。对于基数 p、模 q_0 和自然数 L（选择的层次），令 $q_\ell = p^\ell \cdot q_0$ 对于 $\ell = 1, \ldots, L$。注意，层次 ℓ 的密文是 R_{q_ℓ} 中的一个向量。考虑以下相关分布：对于**实数** σ，$DG(\sigma^2)$ 是一个在 $\mathbb{Z}^d$ 上的离散高斯分布，从独立的离散高斯分布中采样，每个分量的方差为 σ^2。对于 $0 < \rho < 1$ 的实数，分布 $ZO(\rho)$ 是一个在 $\{-1, 0, 1\}^d$ 上的分布。其中，0 的概率为 $1-\rho$，而 -1 或 1 的概率为 $\frac{\rho}{2}$。最后，χ 是一个 B-有界分布。

层次型 CKKS16 方案构造如下：

- **密钥生成：**输入安全参数 λ，选择 M、整数 h 和 t，以及实数 σ，使得对关联的 RLWE 实例，最佳攻击的复杂度为 2^λ。私钥为 $sk = (1, s)$。其中，$s \in \chi$。生成一个均匀随机的 $a \in R_{q_L}$，并计算 $b = -as + e \mod q_L$。其中，e 为 $DG(\sigma^2)$。最后，采样 $a' \in R_{t \cdot q_L}$ 和 $e' \in DG(\sigma^2)$，并计算 $b' = -as + e' + ts' \mod t \cdot q_L$。公钥为 $pk = (b, a)$ 和评估密钥为 $evk = (b', a')$。
- **加密：**消息为 $m \in R$，公钥为 $pk = (b, a) \in R_{q_L}^2$，随机选择 $v \in ZO(1/2)$ 和噪声 $e_0, e_1 \in DG(\sigma^2)$，如下计算密文

$$\mathbf{c} = (\beta, \alpha) = v(b, a) + (m + e_0, e_1) \mod q_\ell = (vb + m + e_0, va + e_1) \mod q_\ell.$$

- **解密：** 输入私钥 $sk = (1, s)$ 和密文 $\mathbf{c} \in R_{q_\ell}^2$，如下计算：

$$\begin{aligned}
&\langle \mathbf{c}, sk \rangle \mod q_\ell \\
&= \beta + \alpha s \mod q_\ell \\
&= vb + m + e_0 + (va + e_1)s \mod q_\ell \\
&= v(-as + e) + m + e_0 + (va + e_1)s \mod q_\ell \\
&= m + ve + e_0 + e_1 s
\end{aligned}$$

- **标量与密文乘法：** 输入标量 k 和密文 $\mathbf{c}$，直接乘

$$k\mathbf{c} = (k\beta, k\alpha) = kv(b, a) + k(m + e_0, e_1) \mod q_\ell = (kvb + km + ke_0, kva + ke_1) \mod q_\ell.$$

- **解密：**

$$\begin{aligned}
&\langle k\mathbf{c}, sk \rangle \mod q_\ell \\
&= k\beta + k\alpha s \mod q_\ell \\
&= kvb + km + ke_0 + k(va + e_1)s \mod q_\ell \\
&= kv(-as + e) + km + ke_0 + k(va + e_1)s \mod q_\ell \\
&= km + kve + ke_0 + ke_1 s
\end{aligned}$$

  解密获得 km。如果 $k = 10000$，则是扩大；如果 $k = 1/10000$ 则是缩小。注意：直接缩小可能会与噪声重叠，解密会出错。通常需要先放大，后缩小，才不会与噪声重叠。

- **同态加法：** $\mathbf{c} + \mathbf{c}' = ((v + v')b + (m + m') + (e_0 + e_0'), (a + a')a + (e_1 + e_1'))$ mod q_ℓ
- **解密：** 输入私钥 $sk = (1, s)$ 和密文 $\mathbf{c} + \mathbf{c}' \in R_{q_\ell}^2$，如下计算：

$$\begin{aligned}
&\langle (\mathbf{c} + \mathbf{c}'), sk \rangle \mod q_\ell \\
&= (\beta + \beta') + (\alpha + \alpha')s \mod q_\ell \\
&= (v + v')b + (m + m') + (e_0 + e_0') + (a + a')as + (e_1 + e_1')s \mod q_\ell \\
&= (v + v')(-as + e) + (m + m') + (e_0 + e_0') + (a + a')as + (e_1 + e_1')s \mod q_\ell \\
&= (m + m') + (v + v')e + (e_0 + e_0') + (e_1 + e_1')s
\end{aligned}$$

- **同态乘法：**

$$\mathbf{c} \cdot \mathbf{c}' = (\beta, \alpha)(\beta', \alpha') = (\beta\beta', \beta\alpha', \alpha\beta', \alpha\alpha') = (d_0, d_1) + t^{-1}d_2 evk \mod q_\ell,$$

  使用**密钥交换**技术使得扩展密文 $\mathbf{c} \cdot \mathbf{c}' = (d_0, d_1, d_2)$ 变成正常密文 $(\bar{\beta}, \bar{\alpha})$。

- **解密：**
  - 如果是扩展密文 $\mathbf{c} \cdot \mathbf{c}' = (d_0, d_1, d_2)$，则使用扩展私钥 $(1, s, s^2)$ 解密；
  - 如果是**再线性化**密文 $(\bar{\beta}, \bar{\alpha})$，则使用正常私钥 $(1, s)$ 解密（通常是该情况，使得加密生成的密文与经过同态计算的密文不可区分）。

需要注意：当我们有两个密文 $\mathbf{c}, \mathbf{c}'$ 处于不同的级别 $\ell' < \ell$ 时，应该通过调整更高级别的密文的级别来匹配两个密文的级别，即 $\ell' = \ell$。这可以通过重新缩放过程实现，该过程将级别为 ℓ 的密文 $\mathbf{c} \in R_{q_\ell}^2$ 转换为 $\mathbf{c}' = \lfloor q_{\ell'}/q_\ell \rceil \mod q_{\ell'}$。

CKKS16 特点：消息空间可以表示为扩展域 C 中的元素。非正式地，消息 m 可以嵌入 $S = R[x]/\langle x^d + 1\rangle$ 中。由于 $x^d + 1$ 的根是复数单位根，若要将 $m \in S$ 转换为复数向量，只需在这些复数根上对其进行求值。第四代方案类似于第二代方案，区别在第四代是**近似计算**，能用于**浮点数计算**，且计算速度显著提高。

3.4.2 CKKS16 应用

CKKS16 实现浮点计算：$1.234 \times 0.689 \times 2.194 \times 0.971 \times 3.323 \times 4.154 \times 0.489 \times 3.772 = ?$

- **ChatGPT 4o:** 46.03
- **Gork 2.0:** 45.8
- **deepseek:** 46.12

逐步计算并保留 2 位小数：$1.23 \times 0.68 \times 2.19 \times 0.91 \times 3.32 \times 4.15 \times 0.48 \times 3.77 = 41.62$

逐步计算并保留所有小数：$1.23 \cdot 0.68 \cdot 2.19 \cdot 0.91 \cdot 3.32 \cdot 4.15 \cdot 0.48 \cdot 3.77 = 41.5594574077313280 \approx 41.56$

先放大，再缩小：$12340 \cdot 6890 \cdot 21940 \cdot 9170 \cdot 33230 \cdot 41540 \cdot 4890 \cdot 37720/10000^8 = 4.361 \cdot 10^{33}/10000^8 = 43.61$

方法：先放大，再计算，再缩小最终结果，则能获得准确值。

消息 1.234 扩大 12340，然后加密，然后同态乘法，然后解密，然后缩小 10000，获得 1.234。加密和解密过程中的噪声对放大的消息影响较小。

缩减技术：CKKS16 的密文能够乘以常量 $1/p$，实现对应明文数据的缩小，用于缩小噪声。

明文	m(1.234)	m(0.689)	m(2.194)	m(0.917)	m(3.323)	m(4.154)	m(0.489)	m(3.772)
起始：乘以放大因子 10000，无噪声								
明文放大	M(12340)	M(6890)	M(21940)	M(9170)	M(33230)	M(41540)	M(4890)	M(37720)
CKKS16 **加密**，引入噪声								
密文	C(12340-3)	C(6890+3)	C(21940+1)	C(9170-3)	C(33230-2)	C(41540+3)	C(4890+2)	C(37720-2)
同态乘法，引入噪声								
密文	C(8503,8941+3)		C(20113,2147-2)		C(138039,0804-2)		C(18451,3256+3)	
密文乘以缩小因子 1/10000，消灭噪声								
密文	C(8503)		C(20133)		C(138039)		C(18451)	
同态乘法，引入噪声								
密文	C(17119,0899-3)				C(254695,7589+4)			
密文乘以缩小因子 1/10000，消灭噪声								
密文	C(17119)				C(254695)			
同态乘法，引入噪声								
密文	C(436012,3705)							
密文乘以缩小因子 1/10000，消灭噪声								
密文	C(436012)							
解密，引入噪声								
明文缩小	M(436012-4)							
最后：乘以缩小因子 10000，消灭噪声								
明文	获得一个结果 m(43.6008)约等于 43.61							

图 9: 近似同态计算

4 社区动态

最初的 Gentry09 FHE 方案比明文计算慢约 1 亿倍，因此被认为远未达到实际应用的标准。经过过去十多年的算法优化，FHE 方案的性能已有大幅提升。从软件的角度来看，FHE 算法库在帮助研究人员和从业者编写第一个基于 FHE 的应用程序方面发挥了关键作用，并且它们的演化和优化显著提高了这些应用程序的效率。然而，利用这些 API 需要对 FHE 方案有深入的了解。近年来，更高层次的工具开始发展，试图弥合开发隐私保护应用程序的工程师与现有技术 FHE 库之间的鸿沟。沿着这一思路，一些 FHE 编译器应运而生，旨在将高级程序转换为基于 FHE 的实现。这些编译器是使 FHE 能够为非专家提供服务的关键步骤，这对于设计隐私保护应用程序的基本构建块至关重要，从而促进了 FHE 的广泛采用。借助编译器，应用程序可采用 Python、C++、Rust 甚至 DSL 等高级语言实现。从硬件角度来看，利用特定的硬件架构（如，基于 FPGA、GPU、CPU、TPU 和 ASIC 等架构），也做了大量努力。这类设计被称为 FHE 加速器，能够在软件中显著提高 FHE 方案的性能。

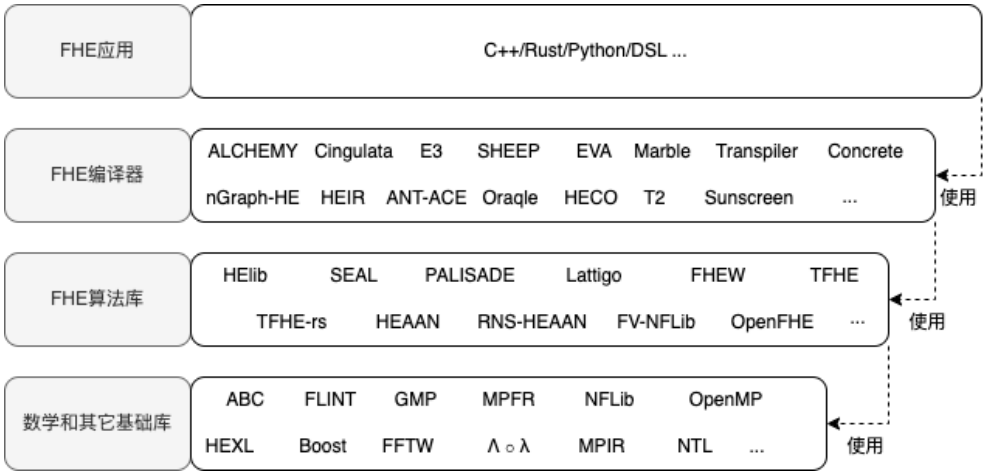

图 10: FHE ecosystem architecture

由上图开源生态展示可知，当前 FHE 编译器创新开发非常活跃，而近年来基础算法库中的理论算法创新仍显不足。

4.1 FHE 开源算法库

FHE 算法库的主要目标是通过 API 提供 FHE 方案的操作。除了由 KeyGen、Enc、Dec 和 Eval 提供的核心功能外，大多数广泛使用的库还包含额外的功能，允许对密文进行维护（即在计算过程中管理噪声增长）和操作，以及同态加法和乘法方法。然而，正确的使用仍然依赖于开发者，开发者必须深入了解每个 API 调用在给定隐私保护解决方案中的含义。

上表中提供了现有的开源 FHE 库、它们使用的编程语言、支持的 FHE 方案以及最后更新的日期：

- 第一个发布的 FHE 算法库是HElib，由 Halevi 和 Shoup 提出，采用 C++ 基于NTL 库实现。HELib 是 IBM 开发的 BGV 和 CKKS 实现。详情见 2014 年论文《Algorithms in helib》。
- SEAL是由微软开发，采用 C++ 和 C# 实现（以支持.NET）。它利用英特尔的HEXL——为启用 AVX-512 的处理器提供高效同态加密操作实现的库。Microsoft 的 SEAL，支持 BGV/BFV 和部分 CKKS，并配有

表 1: FHE 方案开源算法库

Library	Language	BGV	B/FV	FHEW	TFHE	CKKS	Date of last commit
HElib	C++	●	○	○	○	●	18/7/2023
SEAL	C++/C#	●	●	○	○	●	11/7/2024
PALISADE	C++	●	●	●	●	●	19/4/2024
Lattigo	Go	●	●	○	○	●	18/12/2024
FHEW	C++	○	○	●	○	○	30/5/2017
TFHE	C++/C	○	○	○	●	○	26/3/2023
TFHE-rs	Rust	○	○	○	●	○	1/3/2025
HEAAN	C++	○	○	○	○	●	14/8/2023
RNS-HEAAN	C++	○	○	○	○	●	26/10/2018
FV-NFLlib	C++	○	●	○	○	○	26/7/2016
OpenFHE	C++	●	●	●	●	●	30/10/2024

相关的EVA编译器（该项目已不再积极开发）。

- PALISADE是由DARPA 联盟开发的。它是用 C++ 实现的，可配置使用 NTL/GMP 或 tcmalloc 库。
- Lattigo 是由 2020 年论文《Lattigo: A multiparty homomorphic encryption library in go》提出的，是第一个用 Go 实现的库。Lattigo 是纯 Go 语言实现的 CKKS，同时也包括 BGV/BFV 的实现，最初由Jean-Philippe Bossuat开发，目前由Tune Insight维护。
- FHEW由 Ducas 和 Micciancio 提出，用 C++ 实现，但自 2017 年以来未更新。
- TFHE 库由 2019 年论文《TFHE: fast fully homomorphic encryption over the torus》作者提供，用 C++ 和 C 实现，运行需要至少一个快速傅里叶变换（FFT）处理器。TFHE 被认为是 FHEW 库的继任者。TFHE 库在 2021 年 9 月之后不维护了。
- TFHE-rs是 Zama 实现的 TFHE 变体，采用 Rust 实现。TFHE-rs 是领先的 CGGI 实现，采用 Rust 实现，由盈利公司Zama开发和维护。其基于原始的 TFHE 库实现。
- HEAAN库用 C++ 实现，建立在 NTL 库之上。HEAAN 库是经典的 CKKS 实现，当其作者成立了盈利性研究实验室CryptoLab时，该项目转为闭源。最后一个开源版本HEAAN（C++）仍然在 GitHub 上可用，2022 年 1 月之后闭源了。
- RNS-HEAAN用 C++ 实现，但自 2018 年以来未更新。
- FV-NFLlib用 C++ 实现，基于NFLlib C++ 库。NFLlib 是一个专门用于理想格基加密的库，基于数论变换（NTT）。值得注意的是，FV-NFLlib 在过去五年没有更新。
- OpenFHE是一个新库（2022 年 7 月发布），由 PALISADE、HElib、HEAAN 和 FHEW 库的作者设计。它用 C++ 实现，包含所有相关的 FHE 方案：BGV、B/FV、FHEW、TFHE 和 CKKS，还实现了一些 PALISADE 中未覆盖的最近改进。OpenFHE 实现了所有主要的 FHE 方案，包括方案切换，主要由Duality维护。OpenFHE 取代了早期的 PALISADE 项目，PALISADE 项目目前已不再维护。OpenFHE 是唯一支持方案切换的库，并且是DPRIVE项目参与者作为硬件入口点的主要 API。

以上这些 FHE 开源算法库：

- 具有不同的可用性和完成度。
- 适用于单线程 CPU、多线程 CPU、GPU 或 FPGA。
- 每个都有其自身的 FHE 方案实现。
- 每个都有特定的 API 接口。

4.2 FHE 开源编译器

在过去十年中，已有数十种同态加密编译器问世，这些编译器的存在有充分的理由。因为同态加密技术涉及复杂的数学知识，并需要专家知识来将计算表达为同态加密操作。这些编译器为可能没有时间或专业知识手动优化计算的用户提供了支持。另一个原因是，同态加密的计算成本仍然很高，编译器可以帮助用户优化其安全计算任务。

FHE 编译器是旨在抽象化 FHE 库所暴露的技术 API 的高层工具，使得更广泛的开发人员能够安全地实现隐私保护机制。FHE 编译器解决了设计基于 FHE 的应用时常见的工程挑战：

- **参数选择**：为 FHE 方案定义合适的参数值，从而产生安全且高效的实例，并非易事。一些 FHE 编译器允许根据一些预定义的需求自动生成参数。
- **明文编码**：在 FHE 中，明文消息的语义与可以执行的同态计算类型密切相关。某些特定上下文的 FHE 编译器可以用来帮助处理这一问题（如，nGraph-HE）。
- **数据独立执行**：由于 FHE 操作本质上是数据独立的，因此使用 FHE 执行数据依赖的分支步骤并非易事，因为这可能会破坏隐私属性。在这种情况下，可以通过评估两个分支并在最后选择结果的方式来进行分支操作。
- **打包或批处理**：支持将消息打包或批处理到一个单一密文中的 FHE 方案可以直接利用 SIMD 指令集。一些 FHE 编译器已经积极优化了向量化操作。
- **密文维护**：在 FHE 操作过程中，如何优化噪声增长的管理并不简单，FHE 编译器开始使用先进的策略来协助这一传统上复杂的部分。

表2列出了 FHE 编译器，展示了它们使用的编程语言、它们利用的 FHE 算法库，以及它们最后更新的日期：

- ALCHEMY是由 2018 年论文《Alchemy: A language and compiler for homomorphic encryption made easy》作者用 Haskell 实现的编译器。它实现了 BGV，使用了Λ □ λ（发音为 “L O L”）——支持 FHE 的基于环的格加密库。
- Cingulata（以前称为 Armadillo），由 2015 年论文《Armadillo: a compilation chain for privacy preserving applications》作者用 C++ 实现的编译器，建立在FLINT和Sage库之上。
- E3（Encrypt-Everything-Everywhere）是由 2018 年论文《E3: A Framework for Compiling C++ Programs with Encrypted Operands》提出的一个框架，主要用 C++ 实现，并支持多种 FHE 库。
- SHEEP（为 Homomorphic Encryption Evaluation Platform 的递归缩写）是由 Turing Institute 开发的框架，用 C++ 实现，并附带多个现成的 Jupyter notebooks，包含关于如何使用 SHEEP 的示例。
- EVA（Vector Arithmetic Language and Compiler）是由 2020 年论文《EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation》中提出的，用 C++ 实现，并结合了 2019 年论文《CHET: an optimizing compiler for fully-homomorphic neural-network inferencing》来支持 tensor 电

表 2: 公开可用的 FHE 编译器

Compiler	Language	HElib	SEAL	PALISADE	FHEW	TFHE/-rs	OpenFHE	Date of last commit
ALCHEMY	Haskell	○	○	○	○	○	○	15/3/2020
Cingulata	C++	○	●	○	○	●	○	15/1/2024
E3	Pascal/NASL	●	●	●	●	●	○	3/3/2023
SHEEP	C++	●	●	○	○	●	○	7/4/2023
EVA	C++	○	●	○	○	○	○	1/5/2021
Marble	C++	○	●	○	○	○	○	23/12/2020
Transpiler	C++/Starlark	○	●	○	○	○	○	19/3/2024
Concrete	C++/MLIR	○	○	○	○	●	○	19/12/2024
nGraph-HE	C++	○	●	○	○	○	○	3/1/2023
HEIR	C++/MLIR	○	○	○	○	●	●	7/1/2025
ANT-ACE	C++	○	●	○	○	○	●	10/12/2024
Oraqle	Python	●	○	○	○	○	●	27/11/2024
HECO	MLIR	○	●	○	○	○	○	23/7/2024
T2	Java	●	●	●	○	●	○	30/12/2023
Sunscreen	Rust	○	●	○	○	○	○	6/12/2024

路。

- Marble 是由 2018 年论文《Marble: Making fully homomorphic encryption accessible to all》作者实现的 C++ 编译器。
- Transpiler是 Google 主导的 FHE C++ 转译器。该转译器将 Google 的XLS库连接到多个 FHE 后端（目前为 TFHE 库和 OpenFHE 的 BinFHE 库）。详情见 2021 年论文A General Purpose Transpiler for Fully Homomorphic Encryption。已近 10 个月无更新。HEIR 为 Google 的下一代编译器框架。
- Concrete是由 Zama 团队开发的，基于 LLVM 的 TFHE 编译器，可将 python 程序转换为 FHE 等效程序。对应 2020 年论文《CONCRETE: Concrete operates on ciphertexts rapidly by extending TfhE》。
- nGraph-HE由 Intel AI 研究团队 2019 年论文《nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data》中提出，基于该团队 2019 年论文《nGraphHE2: A high-throughput framework for neural network inference on encrypted data》中所提出的 nGraph ML 编译器，后续加入了对非多项式激活函数的支持。nGraph-HE 是特别为 ML 应用设计的领域特定 FHE 编译器。
- HEIR由Google 团队于 2022 年提出，是基于多级中间表示（MLIR）的编译工具链，其他编译器可以重用和扩展它。HEIR 旨在支持所有主要的 FHE 方案和硬件后端。截至目前，它支持将 CGGI 导出到 TFHE-rs（用于 CPU）和 FPT（用于 FPGA），并且正在准备支持将 BGV 导出到 OpenFHE（同时也为 DPRIVE 加速器做准备）。
- ANT-ACE由蚂蚁技术研究院于 2024 年 11 月开源，是专为自动化神经网络（NN）推断而设计的 FHE 编译器框架。ANT-ACE 接受经过预先培训的 ONNX 模型作为输入, 并直接生成 C/C++ 程序以对加密数据执行 NN 推理。ANT-ACE 专为隐私保护机器学习（PPML）推理应用而设计。在此设置中,ML 推

理在云中运行, 使客户端可以上传其数据并接收推理结果。

- Oraqle由 2024 年论文《Oraqle: A Depth-Aware Secure Computation Compiler》提出，通过实现深度感知的算术化方法，同时解决了高级操作的算术化和乘法深度减少的问题。Oraqle 编译器并不专注于单独减少乘法次数或乘法深度，而是同时减少这两者。这样，它生成了多个电路，这些电路在这两个指标之间进行权衡。支持公共子表达式消除（CSE），使用 HElib 库布置同态加密操作，但布置方式相对简单（如，可能会在缩小模数后又重新放大模数）。没有使用 DSL，而是允许用户用纯 Python 表达电路，并通过重载操作符支持 Python 函数的一个子集。
- HECO由 2023 年论文《HECO: Fully Homomorphic Encryption Compiler》提出，类似于 HEIR 编译器，依赖 MLIR 工具链。它支持 Python 前端和 SEAL 后端。它未实现高级原语的算术化，目前这部分任务留给用户完成。HECO 执行简单的参数选择，并采用简单的重线性化操作布置。由于基于 MLIR，它能够执行 CSE，并支持减少乘法深度的矢量化处理。
- T2由 2022 年论文SoK: New Insights into Fully Homomorphic Encryption Libraries via Standardized Benchmarks提出。T2 编译器提供了一种描述算术电路的领域特定语言（DSL），支持为多个库生成代码。如果 p 是素数，该编译器实现了针对等式检查和比较的算术化，但未考虑深度-成本权衡。该编译器从预定义列表中选择参数，并自动布置同态加密操作。支持三种不同的计算模型: 整数、二进制和浮点域。
- Sunscreen编译器由 Suncreen 团队采用 Rust 语言开发，其选择依赖于微软 SEAL 库中的 BFVf 方案，因为BFV 方案在 32 位和 64 位计算方面提供了一些最佳的开箱即用算术运算性能同时专注于自动化参数和密钥选择，并支持序列化和 WASM，使开发人员的设置变得轻松。

此外，HELayers是 IBM 为算术 FHE 设计的编译器，专门用于寻找良好的打包方案。核心代码未开源。目前已用于 IBM 的 FHE 云服务中，使数据科学家和开发人员可以部署隐私在云中保留机器学习驱动的软件即服务（SaaS）应用程序。

HEaaN.mlir 是 CryptoLab 为 HEaaN 设计的编译器，目前未开源。详情见 2023 年论文《HEaaN.MLIR: An Optimizing Compiler for Fast Ring-Based Homomorphic Encryption》。

4.3 FHE 开源加速器

表 3: 公开可用的 FHE 加速器

名称	平台	语言	BFV	CKKS	TFHE/-rs	可编程	可自举	最新提交日期
cuHE	GPU	Cuda/C++	○	○	●	○	●	8/6/2017
CuFHE	GPU	Cuda/C++	○	○	●	○	●	9/2/2019
NuFHE	GPU	Python/Mako	○	○	●	○	●	18/3/2020
HEAT	FPGA	C	●	○	○	○	○	25/8/2020
HEXL	CPU	C++	●	●	○	●	○	19/7/2024
HEXL-FPGA	FPGA	C++	●	●	○	●	○	28/10/2022
HERACLES	CPU	Python	○	●	○	●	○	14/11/2024
FPT	CPU	Rust	○	○	●	●	●	21/8/2024
Jaxite	TPU/GPU	Python	○	○	●	●	●	20/12/2024

表3概述各个开源 FHE 加速器所使用的硬件、开发语言、支持的 FHE 算法方案等信息。其中“可编程”指的是加速器能够支持多种加密参数，而无需重新配置硬件架构。支持“自举”意味着可以执行自举操作，从而实现加速器上无限次的 FHE 操作。”BFV” 代表 BFV/BGV，”TFHE” 代表 TFHE/GSW。对于所有 FHE 加速器，FFT/NTT 模块是其共同的核心组件，因为这是多项式计算的瓶颈，并直接决定了性能。需要指出的是，大多数现有的加速器（除了基于 GPU 的NuFHE ）并未同时实现 FFT 和 NTT 计算，因为 FFT 包含浮点运算，而 NTT 仅包括整数运算。它们需要完全不同的硬件电路进行加速。因此，为简化设计复杂性并减少硬件资源开销，加速器设计者通常选择加速 FFT 或 NTT 中的一种运算，因此只能支持相应运算的 FHE 算法，正如上表所列。通过将 FFT/NTT 模块与其他计算单元以优化的数据路径相连接，加速器可以完成其目标 FHE 工作。不同的工作在 FFT/NTT 模块和数据路径的设计上存在差异，以实现其加速目标。早期的工作由于计算场景有限，通常为 NTT/FFT 计算单元设计了固定的硬件架构和计算流程，以降低设计复杂性。随着应用场景的不断增加，后期工作倾向于设计可编程的 NTT/FFT 单元和数据路径，以支持不同的参数设置和多种应用。此外，后期的研究发现，引入更多带有定制设备的片上存储空间至关重要，因为通用硬件（如 GPU 和 FPGA）的内存已无法满足日益复杂的 FHE 应用所需的大量计算。因此，对于能够负担得起的设计者来说，专用集成电路（ASIC）成为了首选。

- cuHE由 2015 年论文《cuHE: A Homomorphic Encryption Accelerator Library》提出，使用 GPU 进行加速，并提供了 NTT 和 CRT（Chinese Remainder Theorem）的 CUDA 实现。CuFHE于 2018 年提出，cuFHE 利用cuHE的实现来提升 TFHE 的性能。cuHE 和 cuFHE 都是开源的独立加速库不提供与 FHE 库的官方集成。它们仅支持使用 NTT 的多项式运算加速，而基于 FFT 的实现可能在 TFHE 中获得更高的性能。此外，除了功能有限外，cuFHE 采用固定的加密参数，无法配置以实现高通用性。
- NuFHE由 NuCypher 在 2018 年推出，为基于 GPU 的环同态加密实现。nuFHE 提供 FFT 或 NTT 以提高性能，并提供 Python API。通过 FFT，nuFHE 相较于 cuFHE 实现了更高的性能。但它也存在与 cuFHE 类似的通用性限制问题。
- HEAT由 2019 年论文FPGA-Based High-Performance Parallel Architecture for Homomorphic Computing on Encrypted Data提出，与 cuHE、cuFHE 和 nuFHE 不同，HEAT 针对 word-wise FHE 方案 BFV 加速，并进一步利用 FPGA 实现更灵活的硬件设计。因此，HEAT 使用异构的 ARM-FPGA 协同设计架构，并在 Xilinx ZCU102 开发套件和 Amazon AWS 的 f1 实例上做了实现。HEAT 支持的多项式阶数为 4096，无法满足大多数实际需求，极大地限制了其通用性。
- HEXL由 Intel 2021 年论文《Intel HEXL: Accelerating Homomorphic Encryption with Intel AVX512-IFMA52》中提出，与之前依赖于 GPU 和 FPGA 等特定硬件设备的工作不同，利用了 Intel CPU 提供的 SIMD 特性，为 FHE 提供了即插即用的加速能力。HEXL 已通过替换底层算术实现，集成到 PALISADE、Microsoft SEAL 和 HElib 中。由于 HEXL 是线程安全的，用户可以通过多线程并行化不同操作以获得更好的性能。然而，由于架构的不足，当线程数较多时整体性能会显著下降。此外，与其他领域基于 CPU 的加速库类似，HEXL 还面临着内存访问缓慢的问题，这进一步降低了其整体性能。
- HEXL-FPGA由 Intel 于 2021 年提出，为了弥补 HEXL 的缺陷，提供了基于 HLS 的 NTT/iNTT、乘法和密钥切换的实现。用户可以将每个功能编译成单独的比特流并将其加载到 FPGA 设备上。HEXL-FPGA 可以与HEXL集成，以加速相应的 FHE 库。然而，基于 HLS 的解决方案存在固有问题。HLS 设计用高级语言实现，并依赖编译器将代码转换为硬件设计。由于硬件和软件的本质差异，编译过程可能导致

资源消耗冗余和复杂的工作流同步，从而导致性能次优。因此，HEXL-FPGA 很难充分利用 FPGA 的优势。此外，HEXL-FPGA 仅支持有限范围的加密参数。

- Jaxite是 Google 团队 2023 年推出的通过TPU（Tensor Processing Unit）加速 CGGI 的硬件方法。尽管目前性能提升还不显著，但目标是实现 10× 至 100× 的 CPU 性能提升，利用 Google 已大规模部署的 TPU，提前推出 FHE 产品，在更强大的硬件加速器问世前占得先机。

在全同态加密（Fully Homomorphic Encryption，FHE）研究中，一个重要方向是设计定制硬件以加速 FHE 操作。最突出的项目群隶属于 DARPA 的一个计划，称为DPRIVE。简而言之，DARPA 为 FHE 硬件设计者提供资助，挑战他们在 FHE 环境下尽可能快速地完成逻辑回归、卷积神经网络（CNN）推理和 CNN 训练任务。当前参与 DPRIVE 的团队有四个：

- Intel，其加速器名为 HERACLES。具体见 2022 年论文《Intel HERACLES: Homomorphic Encryption Revolutionary Accelerator with Correctness for Learning-oriented End-to-End Solutions》。
- Duality，其加速器名为 TREBUCHET。具体见 2023 年论文《TREBUCHET: Fully Homomorphic Encryption Accelerator for Deep Computation》。该项目目前未开源。
- SRI International，其加速器名为 CraterLake。具体见 2022 年论文《CraterLake: a hardware accelerator for efficient unbounded computation on encrypted data》。该项目目前未开源。
- Niobium Microsystems，其加速器名为 BASALISC。具体见 2022 年论文《BASALISC: Programmable Hardware Accelerator for BGV Fully Homomorphic Encryption》。该项目目前未开源。

所有 DPRIVE 的参与者都在开发加速算术 FHE 方案（如 BFV/BGV 和 CKKS）的专用集成电路（ASIC）。这些项目正处于不同的制造阶段，初步性能数据基于仿真结果。如，Niobium 的初始论文声称，其加速器在处理一个包含 1,024 个样本和 10 个特征的数据集上的逻辑回归任务时，能达到约 5,000× 的性能提升：硬件耗时约 40 秒，而 CPU 需要 60 小时。而这只是硬件加速潜力的下限，未来性能将更高。

这些加速器的核心是对数论变换（Number-Theoretic Transform，NTT）及相关多项式环运算的加速。其主要难点在于：集成足够的 RAM 来存储所有密文及辅助密钥材料；优化内存局部性以提高性能。

鲁汶大学 COSIC 研究实验室开发的基于 FPGA 的方法，名为 FPT（详情见 2022 年论文《FPT: a Fixed-Point Accelerator for Torus Fully Homomorphic Encryption》）。该项目使用 Alveo U280 FPGA，以流水线方式处理 16 个密文的批量，自举函数的吞吐量达到 1 次自举/35 微秒。其曾现场演示过，在 CGGI 下运行 Conway's Game of Life，动画几乎是实时的。

与 DPRIVE 中以 NTT 优化为主的硬件不同，FPT 是一种专注于快速傅里叶变换（FFT）的硬件实现。该项目以 TFHE-rs 的 API 为基础，专门针对 CGGI 设计。此外Optalysys公司正在开发一种光学计算芯片，利用光通过透镜（或纳米级等效物）时的干涉模式，实现“光速”计算傅里叶变换，从而加速自举过程。

2024 年 10 月，全同态加密硬件技术联盟（FHE Technical Consortium for Hardware，FHETCH）成立，旨在提高全同态加密 (FHE) 硬件和软件之间的互操作性，以实现实际应用。该联盟由Optalysys、Chain Reaction和Niobium创立，将汇集 FHE 软件开发商、硬件制造商和云提供商，以加速开发具有商业可行性的 FHE 解决方案。FHETCH 致力于缩小当前 FHE 解决方案与实际商业部署需求之间的性能差距。该联盟将专注于开发 FHE 硬件加速技术，创建开放的技术标准，确保 FHE 系统符合全球监管框架并与现有数据中心格式和 FHE 软件库无缝集成。这些努力将推动 FHE 的商业可行性，降低采用门槛，并扩大其在各个行业的应用。

5 FHE+ 区块链前沿

区块链的一个主要问题是所有交易和数据都是公开可见的，这对需要使用敏感信息（如个人或财务数据）的应用程序开发人员来说是一个挑战。

FHE 可解决区块链的计算机密性、数据安全、用户报名、法规遵从性以及智能合约隐私问题，用于构建链上机密支付、机密投票、保密投票、机密智能合约、盲拍、trustless 游戏、trustless 桥、机密机器学习、去中心化身份、机构金融等。

目前以太坊社区PSE Acceleration Program，致力于赋能 ZKP、FHE、MPC 及相关技术领域的个人，目前有多个 FHE 相关提案。

也涌现了很多公司和团体，致力于提供与区块链兼容的 FHE 解决方案，如：Zama 和 Sunscreen 等公司致力于提供用于构建机密智能合约的工具和库等；OpenFHE 这样的库开始提供面向区块链生态系统的功能（如 OpenFHE 的新 Rust wrapper）；以及大量基于 FHE 开发的链或应用程序项目。

目前来看，除 Octra 使用了 HFHE 算法，其它项目多使用现有的 FHE 算法，且大多数项目是基于 Zama 提供的工具和库来构建应用。但现实情况是，IBM、微软、Google 等巨头布局 FHE 生态多年，若其切入到区块链领域，实力不容小觑。

5.1 Zama：FHE 解决方案提供商

Zama是一家开源加密公司，致力于为区块链和 AI 构建最先进的 FHE 解决方案。由 Rand Hindi 和著名密码学家 Pascal Paillier 于 2020 年初共同创立，Zama 提供用于 Web2 和 Web3 项目的 FHE 解决方案，如 TFHE-rs 库、TFHE 编译器 Concrete、隐私保护机器学习 Concrete ML 和机密智能合约 fhEVM。其源代码更新非常活跃。需注意Zama 技术用于商用目的时，需获得 Zama 专利授权后才能使用。

2024 年 3 月 7 日，Zama 获得了7300 万美元的 A 轮融资。

Zama 创建的fhEVM，是一种机密的智能合约解决方案，使开发人员能够使用 Solidity 创建机密的链上应用程序。fhEVM 可以集成到任何与 EVM 兼容的链中，确保从提交交易到智能合约处理交易的那一刻，状态始终保持端到端加密。

2024 年 12 月 6 日，Zama 团队发布 fhEVM 协处理器，使得可在以太坊、Base 和其他 EVM 链上运行 FHE 智能合约，允许开发人员在任何 EVM 链上构建机密智能合约，包括那些原生不支持 FHE 的链，而无需对底层协议进行任何更改。其 tps 速度由一年前的每秒 2 笔，提高到了每秒近 20 笔，速度提升了 10 倍，且现有新架构具有很高的可扩展性，只需添加更多硬件，就有希望能实现每秒数百甚至数千笔 FHE 交易。

5.2 Sunscreen：FHE 隐私计算引擎

Sunscreen专注于 Web3 隐私解决方案，将 ZKP 与 FHE 结合。所构建的 FHE 编译器，支持将普通 Rust 函数转换为具有隐私性的 FHE 等效函数，当前支持 BFV 以及 TFHE 等同态加密方案，并提供了FHE 编译器 playground。所构建的ZKP 编译器，致力于面向希望同时使用 FHE 和 ZKP 的开发人员，当前支持将 Bulletproofs 作为 ZKP 后端。其源代码近期更新较活跃。

2022.7，Sunscreen 获得由 Polychain Capital 领投的 465 万美元种子融资。2023.9 发布了私有测试网。

5.3 Fhenix：以太坊 L2 FHE + Optimistic Rollup

Fhenix 定位为以太坊的 FHE + Optimistc Rollup，其完全兼容 EVM 和 Solidity，使用 FHE 实现链上机密智能合约。Fhenix 底层采用 Arbitrum Nitro 和 Zama 的 FHE 方案。2024 年 6 月，获得由 Hack VC 牵头的 1500 万美元的 A 轮融资，共筹集了 2200 万美金。开源代码库见：https://github.com/fhenixprotocol，代码更新活跃。

2024 年 11 月 20 日升级至 Fhenix Nitrogen 测试网，最大特点是引入了 Threshold Network 来实现去中心化、信任最小化的解密操作；同时为解决因 FHE 密文很大导致以太坊链上存储成本高的问题，引入了 Celestia 作为数据可用层，来降低交易成本。

目前Fhenix 测试网平均产块间隔为约 4 分钟，日均交易约 1 千笔左右。

5.4 Inco：模块化隐私 L1 链

Inco 定位为模块化机密计算 L1 区块链，用作现有 EVM 和 Cosmos L1 链的通用隐私层。底层为 Cosmos 和 Zama 的 fhEVM。其源代码近期代码更新不太活跃。

2024 年 2 月上线 Inco Gentry 测试网，近期该测试网没有交易。2024 年初获得了由 1kx 领投的 450 万美元种子融资。

5.5 SherLOCKED：ZK+FHE+MPC 隐私基础设施

SherLOCKED定位为 EVM 链的基于 FHE 的隐私基础设施，为 EthOnline'23 黑客马拉松决赛入围者。其源代码近两年代码无更新。

开发人员可使用 SherLOCKED 实现自定义智能合约，通过 sherLOCKED SDK 构建加密交易以及解密加密数据，节点网络执行 MPC 管理解密私钥，利用 RISC0 zkVM 来证明 token 余额的加法和减法同态运算。

5.6 FRAMED!：链上隐藏游戏

FRAMED!为使用 fhEVM 构建的完全链上且无需信任的隐藏信息游戏（黑手党），为 ETHGlobal NYC 2023 决赛入围者。FRAMED! 能够直接在链上存储加密状态。支持 FHE 的智能合约允许对加密数据进行计算，而无需解密，其核心游戏逻辑在 Inco 测试网上作为全同态智能合约实现。其源代码（TypeScript），2024 年 4 月 4 日之后代码无更新。

5.7 Privasea：FHEML 推理网络

Privasea定位为 FHEML 推理网络，基于 Zama concrete 技术。

2025 年 1 月 10 日，Privasea 宣布以 1.8 亿估值获得了 1500 万美金的 A 轮融资。

尽管其公开源代码自 2023 年 7 月份之后无更新，但其发布了多款基于 FHE 的应用，如 Privasea DeepSea AI 网络致力于推进人机协作和智能代理生态系统，相应测试网 Beta 版将于 2025 年 1 月 27 日上线，并计划于 2025 年一季度上线主网；ImHuman为基于 FHEML 的 Dapp，已上线 Google Play 和 App Store；BotOr_NotABot telegram 插件，通过 OTP 验证将 Telegram 帐户链接到 ImHuman 应用程序，提供无缝的人工验证，同时确保用户的敏感生物特征数据安全。

5.8 Octra：基于超图的 FHE L1

Octra是一个支持隔离执行环境的 FHE 区块链网络，提出了一种称为 HFHE（FHE on Hypergraphs）的新型 FHE，并声称 HFHE 可以与任何项目兼容并独立运行。2024 年 9 月获得400 万美元的 pree-seed 融资。2024 年 9 月发布公开 WASM sandbox，展示了 FHE 中 hypergraphs 的高效性。目前可在 HFHE-WASM 沙盒中测试加法和减法，未来将持续添加更多功能。其源代码主要采用 OCaml 语言实现。其核心为基于超图的 FHE 库：https://github.com/octra-labs/HFHE，未能找到相应论文，只有一个简单的算法说明。HFHE 算法相对较新，学术讨论有限。解决方案的安全性尚未验证，需要进一步验证。

2025 年 1 月 7 日发布了Octra 测试网 phase 1，有超过 300 个独立 validator，需要 1T 内存，1T 硬盘。

5.9 Mind Network：FHE 重质押层

Mind Network定位为 PoS 和 AI 网络的 FHE 重质押层，由 Zama 支持。其源代码近半年无代码更新。

2024 年 9 月，获得1000 万美金 Pre-A 轮融资。2024 年 4 月发布激励测试网活动，2024 年 5 月活动结束。其 FHE Bridge 是首个面向受监管金融机构的量子抗性 FHE 桥，用于确保跨链交易的安全。已与 SWIFT 和几家全球银行进行了试点。与 Chainlink 签署了一级渠道合作协议。该产品基于以太坊基金会 2023 年研究成果FHE-DKSAP: Fully Homomorphic Encryption based Dual Key Stealth Address Protocol。

5.10 Co-FHE：基于 Wasm 的 FHE

Co-FHE 背后团队信息不详，其源代码，近半年无代码更新，其核心开发者维护了以下 4 个库：

- https://github.com/Co-FHE/fhe-wasm（WebAssembly）：基于 FHE 的 Wasm 模拟器。
- https://github.com/Co-FHE/wasmi（Rust）：fork 自 Wasmi Labs 的 wasmi 库，定位为 Wasm 解析器。
- https://github.com/Co-FHE/tfhe-rs（Rust）：fork 自 Zama TFHE-rs 库，为 FHE 算法库。
- https://github.com/Co-FHE/zkrpc：基于 Zcash Halo2 实现 RPC ZK 证明。

5.11 Verisense Network：FHE VaaS 网络

Verisense Network定位为由 FHE 赋能的 VaaS（Validation-as-a-Service）网络，可与任意重质押层配合使用，解锁 AVS（Actively Validated Service）需求。其源代码，代码更新活跃。

其核心论文主要有 2 个：

- 2024 年 8 月Monadring：为借助 VRF 和 FHE 实现的轻量级共识协议。
- 2024 年 8 月Verisense 白皮书。

计划于2025 年 Q1 上线测试网。

5.12 Airchains：zkFHE L2

Airchains：借助 Zama fhEVM，正在重新定义机密性和透明度如何在链上共存，其强大的框架将 zk 和 FHE 结合在一起的 L2。技术白皮书为Leveraging Zero-Knowledge and Fully Homomorphic Encryption Technologies。目前处于测试网阶段。其源代码，近期代码有更新。

5.13 Fair Math

Fair Math定位为去中心化 FHE 计算机，致力于成为下一代安全和去中心化应用程序的基础平台。其目标是构建一个可扩展的开源生态系统，使隐私保护计算能够广泛应用。通过克服与隐私和机密性相关的障碍，解锁新的用例并推动创新。其源代码，近期更新不太活跃。

2023 年底，Fair Math 和 OpenFHE 社区一起上线了 FHERMA 平台，详细信息见 2024 年初论文《FHERMA: Building the Open-Source FHE Components Library for Practical Use》，其引入了黑盒和白盒挑战，并对提交的 FHE 解决方案进行了全自动评估。FHERMA 平台上的最初挑战是由机器学习和区块链中的实际问题引起的。获胜的解决方案被集成到 FHE 组件的开源库中，该库在 Apache 2.0 许可证下可供 PET（Privacy-Enhancing Technology）社区的所有成员使用。目前FHERMA 平台上已发布了多个挑战。2024 年 2 月获得140 万美金 Pre-seed 轮融资。2024 年 11 月发布了技术白皮书《Decentralized FHE Computer》。

5.14 FHE 方案评估

目前公开维护活跃的 FHE 算法库只有：

- SEAL（C++/C#）：主要由微软维护。
- Lattigo（Go）：主要由Tune Insight维护。Tune Insight 是一家诞生于瑞士洛桑 EPFL 数据安全实验室的初创公司。
- TFHE-rs（Rust）：主要由 Zama 维护。
- OpenFHE（C++）：主要由Duality维护。Duality 由世界知名的密码学家和数据科学家专家创立，旨在帮助组织与其业务生态系统（客户、供应商和合作伙伴）安全地协作处理敏感数据。

这些库代码实现上没啥明显缺陷，都是实现的多年前的 FHE 理论算法。不过 Zama 背后采用的 TFHE 算法，其本质不支持浮点运算，若用于机器学习等场景，其Concrete 框架是做了一些额外处理的。若考虑 AI 场景应用，Zama 的框架不一定是最优的，不过目前其整个套件支持应该是对应用项目方最友好的。

FHE+ 区块链项目中，除 Zama、Sunscreen 团队有很好的 FHE 理论背景外，Octra（其 HFHE 无详细论文，实力不详），其它项目都是偏应用，且几乎都是用的 Zama 提供的框架。

目前比较权威的性能评测有：

- 2022 年论文《SoK: New Insights into Fully Homomorphic Encryption Libraries via Standardized Benchmarks》。距今已过去 2 年多的时间了，各个库可能有迭代优化，各个指标应该是不准确了。
- HE Benchmarking Framework - HEBench：2023 年 8 月之后再无更新。
- 2024 年论文《Benchmarking Fully Homomorphic Encryption Libraries in IoT Devices》。针对的是资源受限的 IoT 设备上的评测。

性能评测其实很难有统一标准，各个库可能支持不同的加速器（CPU、GPU、FPGA、TPU、ASIC），以及项目方实际项目运行环境也会有不同限制，建议针对自身实际情况来针对性的评测。

另外，建议实际项目选型时除考虑框架友好度之外，可以结合 TFHE-rs（Rust）、SEAL（C++/C#）、Lattigo（Go）、OpenFHE（C++）来综合评估，选择最适合的基础库。

5.15 FHE 方案潜力股

目前来看，整个 FHE 生态正在成熟，社区和标准化也正在形成。目前由政府和工业联盟组织的Homomorphic Encryption Standardization专注于 FHE 的 ISO 标准化工作，同时还举办FHE 研讨会。由 Zama 领导的FHE.org 小组每周会举办一次关于 FHE 研究的研讨会，有一个活跃的Discord，且每年会举办FHE 年度会议。OpenFHE 项目有一个活跃的FHE Discourse论坛，里面有关于 OpenFHE 和 FHE 的讨论。TFHE-rs、Sunscreen和OpenFHE等开源生态更新活跃。DARPA 的DPRIVE项目专注于 FHE 硬件加速，过去 4 年来为 FHE 硬件设计者提供了超过 7 千万美金的资助。IDASH每年会组织隐私和安全研讨会，并发布相关竞赛任务，为生物领域的同态加密做基准测试和加速。

对于区块链领域，目前 Zama 的先发优势明显，其开发的用于机密智能合约的 fhEVM 库，已被很多项目采用。Sunscreen 团队正在迎头赶上。从整个 FHE+ 区块链生态繁荣发展的角度来看，需要有更多的团队参与 FHE 理论算法创新、FHE 算法库多语言开源实现、FHE 加速、FHE 编译器、FHE 各种应用配套 SDK 工具等。

FHE 应用在实际做方案选型时，应考虑所需计算次数、明文空间、密钥大小、密文大小、性能、是否对不同明文或密文做相同操作、开发者友好程度等因素综合考虑，来选择适合该应用的最佳 FHE 方案。目前来看，对比 ZKP 发展轨迹，目前仍处于 FHE 的早期阶段，后续应该会有类似的 FHE 算法寒纪武大爆发，从而能更好的满足实际应用落地场景的性能、安全等要求。

6 FHE 与 ZKP 融合

隐私问题一直是一个需要解决的关键领域，不同的密码原语提供不同的隐私效用：

- ZKP：用户能够证明自己知道某个值，而无需透露该值。可保证计算完整性，但需要知悉数据的明文信息，来证明基于这些数据的计算是正确的，对于某些特定隐私保护应用场景，是明令禁止提供数据明文信息的。
- FHE：允许任何人直接对加密数据进行计算；结果与在纯文本上运行相同。密文的可塑性是 FHE 的一个定义性特性，因为它是允许在加密数据上执行同态计算的必要条件。这意味着 FHE 自身无法保证该计算的正确性，因此也无法提供任何计算完整性保证，使得在真实世界的 FHE 部署时，存在可疑的隐私攻击问题。

FHE 与 ZKP 融合存在 2 个维度：

- 从区块链应用领域角度来看，将ZKP 与 FHE、智能合约结合，可解决新的不可能三角，其中智能合约提供可编程性、ZK 提供可扩展性，而 FHE 提供隐私性。
- 从 FHE 算法自身的角度来看，由于 FHE 密文固有的可塑性，其天生缺乏完整性，即当存在可能以任意方式恶意偏离协议的主动攻击者时，其不仅会影响正确性，还可能完全破坏 FHE 的机密性安全保证。现有的大多数 FHE 工作都依赖于半诚实服务器假设，服务器不会偏离预期计算。而借助 ZKP 技术，服务器正常计算 FHE 电路并存储所有（加密的）中间结果，然后计算 ZKP proof 证明其知道中间值的赋值，使得对于给定输入，电路输出为目标密文。从而可降低对服务器信任假设，提供更强大的完整性保证。同时也可借助 ZKP 来验证密文或密钥的有效性，确保其未被篡改或损坏，防止攻击并保证数据

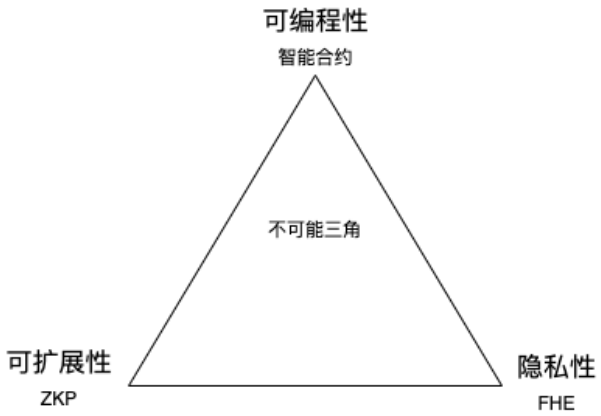

图 11: ZKP、FHE 以及智能合约结合

完整性。

在无信任环境中构建支持 FHE 的应用程序时，ZKP 非常有用且通常是必需的。在此所说的无信任是指以下两种情况之一：

- 情况一；无法完全信任提供 FHE 加密数据的用户或
- 情况二：无法完全信任负责对加密数据执行计算的第三方。

具体场景为：

- 对于情况一，假设用户想要从她的加密账户余额 ‘enc(bal)‘中提取一些加密金额 ‘enc(amt)‘。用户可能会（无意或恶意地）尝试发送比她实际拥有的更多的钱（使得 ‘amt > bal‘）。如何实现 ‘amt <= bal‘在将这两个值保密的同时保证其他方无法看到？使用 ZKP，用户可以证明 ‘amt <= bal‘而无需向任何人透露这些值实际上是什么。**用户提供的密文也必须“格式正确”——这意味着它们符合底层加密方案的要求，而不仅仅是随机的数据字符串**。
  - 如果不信任用户，如何确保确实如此？答案是使用 ZKP！用户可随密文提供 ZKP，证明密文的有效性，而无需透露任何底层值。
- 对于情况二，考虑 web3 中的工作原理，其中的格言是 “don’t trust, verify 不要信任，要验证”。如何知道第三方运行了所要求的 FHE 计算？答案是使用 ZKP！计算方必须证明他们忠实地执行了 FHE 计算。可将这种设置称为 “vFHE（可验证的 FHE）”。与 rollup 所需的有效性证明类似，计算方必须证明他们忠实地执行了 FHE 计算。将这种设置称为 “可验证的 FHE”（通常通过 ZKP，可以在其中验证 FHE 计算是否正确执行）。

6.1 用 ZKP 证明”FHE 执行”

目前已对用 ZKP 证明”FHE 执行” 做了多种尝试：

- zkFHE团队 2023 年所做的Arithmetizing FHE in Circom尝试表明，使用 Circom 算术化 TFHE 中的单个自举 NAND 门，需要约 15 亿个约束；使用 Circom 算术化 FHEW 中的单个自举 NAND 门，需要约 25 亿个约束。尽管获得的约束总数对于任何实际应用而言都不切实际——证明 “即使是简单的 FHE 计算

的正确执行”（如验证单个同态加法和同态乘法运算）也需要几分钟的时间，但按操作划分的约束数量有助于识别出 FHEW 和 TFHE 方案中在零知识证明中需要更多努力的操作。为此 zkFHE 团队认为，应该由专门定制的 FHE-friendly ZKP 系统来证明 FHE。尽管 Circom 支持多种证明后端，circomlib-FHE 的设计目的并不是直接用于证明和验证加密计算。相反，它是一个原型工具，可快速、高效地测试和优化 FHE 方案在证明系统中的算术化表达。

- 2023 年，L2IV Research 团队和https://github.com/emilianobonassi/zkFHE均尝试了用 RISC0 zkVM 来验证 FHE，性能并不理想。
- zkFHE团队正在构建https://github.com/zkFHE/zkOpenFHE，尝试用libsnark来证明OpenFHE 库中 FHE 电路的正确评估，实现了约束的自动生成优化以及扩展 witness 的计算。

当前最先进的 ZKP 证明系统效率极高，但主要针对与 FHE 共享特征较少的应用进行优化。表达现代最先进的 FHE 方案面临一系列挑战，不仅因为同态操作，尤其是密文维护操作计算复杂，还因为 FHE 和证明系统在本质上操作于不同类型的代数结构上。

将 FHE 方案做 ZK 算术化证明时，需要克服的一些挑战：

- FHE 方案使用的环（Z_q 和 $Z_Q[X]/(X^N+1)$）与 R1CS 的有限域 F_p 之间的匹配问题。
- 模数切换中的舍入。
- 不同基数的多位分解和符号分解。
- 数组索引访问动态未知，如在密钥交换或 FHEW 的累加器更新中。
- 大量使用数论变换（NTT）。
- 大规模的 RLWE-RGSW 乘法和多个模约减操作。

FHE 依赖于基于格的密码学，而 web3 空间中使用的最有效的 ZKP 则不然。为了证明 FHE 密文是格式正确的，需要证明它满足某个矩阵向量方程，其中的条目来自特定的数学“空间”（称为环），并且这些条目是“小的”。证明这些条目是“小的”是零知识领域中最昂贵的部分（通常有数千个 range proofs）。2023 年论文《Verifiable Fully Homomorphic Encryption》中，对 Bulletproofs、Aurora、Groth16、Rinocchio 等证明系统的 vFHE 实现性能进行了对比，事实证明，即使使用 Groth16 等高效证明系统，证明“简单的 FHE 计算的正确执行”（如验证单个同态加法和同态乘法运算）也需要几分钟的时间。

其中 Bulletproofs、Aurora、Groth16 是现有通用 ZKP 证明系统，而 Rinocchio 是支持 FHE 的证明系统，由 2021 年论文《Rinocchio: SNARKs for Ring Arithmetic》中提出，但其仅支持算术环操作，使用了一种高度低效的编码系统，表达能力有限，只能够表达现代 FHE 方案中的部分内容。zkFHE 采用了新的编码方式对 Rinocchio 进行了优化，相应代码实现见：https://github.com/zkFHE/ringSNARK（C++），但性能仍不够理想。

未来可能需要 FHE-friendly ZKP 证明系统：如 2024 年 7 月 Dan Boneh 和 Binyi Chen 论文《LatticeFold: A Lattice-based Folding Scheme and its Applications to Succinct Proof Systems》中所提出的 LatticeFold 证明系统，其在全同态加密（FHE）和格签名方案使用的相同模结构上运行，且可使用这种基于格的证明系统来更好地证明关于基于格的关系的陈述，不久的未来应该能看到将用于证明 FHE 具体性能数据。

6.2 用 ZKP 证明”FHE 参与方的 RLWE 密文格式正确”

在无需信任的环境中使用 FHE 时，用户通常必须：加密她的输入并证明这些输入密文是“格式良好的”。

以太坊基金会 2024 年论文Greco: Fast Zero-Knowledge Proofs for Valid FHE RLWE Ciphertexts Formation》，使用 ZKP 来证明 FHE 参与方的 RLWE 密文格式正确，开源代码见：

- https://github.com/privacy-scaling-explorations/greco（Rust & Python）【基于 Halo2 证明系统】：证明 BFV 全同态加密方案中加密密文的正确性。
- (https://github.com/enricobottazzi/zk-fhe（Rust）【基于 Halo2 证明系统】：使用 ZK 来证明，BFV 全同态加密方案中，加密运算的正确执行。

在 Multi-party FHE 应用中，各方对自己的秘密数据加密，并将加密后的密文提交给 server，根据应用逻辑，server 会对各方所提交的密文做同态运算。

如，在不记名投票中，会将编码了选票的密文求和来计数。有效的加密选票形式为 $E(0)$ 和 $E(1)$。恶意投票人可能会发送无效的加密选票，如 $E(145127835)$，从而搞乱整个选举。因此，选民必须证明其提交的密文是：一个有效 RLWE 密文，且，其所加密的明文信息对应有效选票（如要么为 0，要么为 1）。

Greco 用 zero-knowledge proof 让用户证明其 RLWE 密文是 well-formed（符合规则的），即相应的加密操作是正确的。Greco 生成的 proof，可与其他额外的特定应用逻辑组合，并在非交互配置中可被公开验证。

6.3 用 ZKP 证明"FHE Bootstrapping 的正确执行"

Bootstrapping 操作通常为 FHE 方案中最昂贵的操作，尽管 TFHE 的 Bootstrapping，比其它 FHE 方案中的 Bootstrapping，计算开销更低，但其也足够复杂，使得若直观对其做 SNARKifying 操作，将导致 Prover 需要巨大的内存，对大多数应用来说是不切实际的。直接对 TFHE 的 Bootstrapping 做 SNARK-ifying 操作，不是高效的方案。Zama 团队曾实践过，使用 FHE toy 参数（不是真实参数），在具有 128GB RAM 的 AWS 实例上运行，直接内存溢出。

Zama 团队 2024 年论文Towards Verifiable FHE in Practice: Proving Correct Execution of TFHE's Bootstrapping using plonky2中，首次阐述了，在实践中，将整个 FHE Bootstrapping 操作，使用 SNARK 来证明。该论文方案充分利用了 TFHE Bootstrapping 操作中的结构化特性：TFHE 的 PBS 中，包含一个具有多次（约 600 到 700 次）迭代的循环；SNARKs 技术可使用Incrementally Verifiable Computation（IVC）来高效证明该循环，而不是将该循环展开为一个大的电路。所谓 IVC，是指一次证明一个迭代，而不是一次证明所有迭代。同时对其 TFHE 进行了少量调整，使其更适合于基于有限域的算术电路模型，如：

在其开源代码https://github.com/zama-ai/verifiable-fhe-paper（Rust，Plonky2）中，使用 Plonky2 实现了基于递归的 IVC 方案，并将该 IVC 方案用于了 TFHE PBS。

Zama 团队对其 TFHE 进行了少量调整，使其更适合于基于有限域的算术电路。如：将密文的 modulus，修改为 Plonky2 原生使用的素数；修改了 key switch，使得可通过 external product 来执行。借助基于 IVC 的实现，所需内存量降了一个量级，使得消费级笔记本也可运行该 Prover。在经典 AWS 示例中，计算 proof 需约 20 分钟，仍相对较长，但已接近实用水平了。proof size 约为 200kB，验证 proof 时长仅需要数毫秒。

因此，Zama 团队认为：vFHE 处于实用化的边缘。证明 FHE 方案中最困难部分（Bootstrapping），可使用具有非常合理计算资源的 SNARK 来证明。这是一个重要的里程碑，也是一个开始。仍有许多途径待探索，如：如何将 Prover，由 PBS 电路，扩展到，完整的 FHE 电路？是否有改进基于通用 zkVM 的实现的技术？是否有更适合 FHE 的 zkVM？使用基于 folding 的 IVC（如见 2024 年论文《Mangrove: A Scalable Framework for Folding-based SNARKs》），而不是基于递归的 IVC，能否获得更好的效果？Zama 团队坚信 vFHE 方案将

对隐私技术产生巨大影响，并将解锁众多惠及所有人的应用程序。

7 未来方向

尽管同态加密（FHE）在支持加密数据的安全计算方面潜力巨大，但其广泛采用仍面临诸多障碍。FHE 面临的一些主要挑战有：

- 计算复杂性：与 ZK 相比，FHE 计算量明显更大，速度更慢，使得它在某些实际应用中不太实用。
- 密文尺寸庞大：同态加密通常导致密文扩展，即加密数据的尺寸远大于原始明文。这增加了存储需求和通信开销，特别是在处理大规模数据集时。
- 噪声累积：同态加密方案在执行同态运算时容易出现噪声累积，可能降低加密数据的质量并影响计算的准确性。在保护数据隐私的同时管理和减少噪声是一项重大挑战。
- 功能受限：尽管 FHE 支持对加密数据进行任意计算，但在高效执行某些操作和计算类型上存在实际限制，如 RAM（随机访问模型）计算等。
- 密钥管理与分发：FHE 需要生成和管理加密密钥，包括公钥和私钥，这使密钥管理和分发过程更加复杂。在确保高效访问控制的同时，安全分发和保护密钥是一项非平凡的任务。
- 密钥管理的复杂性以及对有效噪声管理的需求进一步增加了它的局限性。
- 标准化与互操作性：缺乏标准化的 FHE 方案和可互操作的实现，阻碍了不同平台和环境间的协作和采用。推动标准化对于促进互操作性并确保不同 FHE 实现间的兼容性至关重要。

为此需要：

- 改进算法：需要持续开发更高效的算法和技术，以降低 FHE 方案的计算复杂性并提升其性能。这包括优化同态运算、最小化噪声累积以及改进加密和解密算法。
- 参数选择优化：选择合适的加密参数并优化方案参数，对于在 FHE 实现中实现更好的性能和安全性至关重要。参数选择技术和优化策略旨在平衡性能、安全性和功能性。
- 硬件加速：专用硬件加速技术（如 FPGA、ASIC 等加密处理器）正在探索中，以加速同态计算并减少延迟。基于硬件的解决方案可显著提升 FHE 在特定应用和使用场景中的效率。目前已有一些在研的 FHE 加速器，详情参加本文“FHE 开源加速器”章节内容。
- 混合加密方法：结合 FHE 与其他加密技术（如对称密钥加密 AES），旨在利用两种方案的优点，同时减轻各自的局限性。通过仅使用 FHE 加密部分数据，混合方法能够减少计算开销和存储需求。
- 标准化工作：行业联盟和标准化机构正致力于开发 FHE 的通用标准和协议，促进跨平台和应用的互操作性与采用。标准化工作推动了 FHE 实现间的协作、互操作性和兼容性。

从 FHE 工程落地角度来看，期待有：

- 1) Threshold FHE（门限 FHE）：目前 FHE 区块链项目大多使用一个通用公钥进行加密，使用分布式私钥进行解密。主流的门限解密方案不支持去中心化的 validator 集。@dWalletLabs在其 2024 年《2PC-MPC: Emulating Two Party ECDSA in Large-Scale MPC》白皮书中将 FHE 与 MPC 结合，致力于实现信任最小化隐私的飞跃。

- 2）Multi-key FHE（多密钥 FHE）：门限 FHE 方案中经常未解决的一个问题是通用解密密钥只能解密使用通用加密密钥加密的密文。拥有通用密钥会在隐私保证方面带来一些严重的权衡。多密钥 FHE 通过对使用多个密钥加密的数据进行直接计算来规避这个问题，产生的结果只有在所有相关方（即用户和 validators 验证者）的同意和合作下才能解密。该方案未被利用的主要原因是它在本已繁琐的方案上产生了巨大的计算开销。
- 3）Verifiable FHE（vFHE）：FHE 的另一个发展领域是可验证计算以及 FHE 完整性验证。可能需要有 FHE-friendly ZKP 方案。
- 4）新方案和实现：如今，格统治着 FHE 密文的世界。希望能有实现 FHE 方案的全新方法。正如 Craig Gentry 在其 2022 年论文《Homomorphic encryption: a mathematical survey》中所指出，在 FHE 算法理论创新方面，未来有可能：能构建“无噪声”的 FHE；能实现针对量子电路的 FHE 等。
- 5）加速：让 FHE 性能能满足实际应用需求。
- 6）FHE 编译器：更好的 FHE 编译器，能助力开发者体验，提升 FHE 应用落地进度。

8 前景展望

在过去几年中，FHE 的性能取得了多次飞跃，使其成为多种安全计算应用的有力解决方案。研究重点已从展示概念验证应用转向解决实际问题，目前已经看到第一波基于 FHE 的实际应用部署，如苹果公司最近在 iOS 上引入了基于 FHE 的实时来电显示功能，详情可参看苹果 2024 年 7 月博客Announcing Swift Homomorphic Encryption以及 2024 年 10 月博客苹果生态的机器学习和同态加密。

Data Bridge Market Research 分析了全球全同态加密市场，该市场在 2023 年为 2.9762 亿美元，预计到 2031 年将达到 5.5087 亿美元，在 2024-2031 年预测期内的复合年增长率为 8.00%。随着 FHE 理论算法和工程实现的快速发展，实际表现应能远超该预测值。

将 ZK 推向今天的水平总共花费了大约 10 亿美元——这是一个很大的兴奋和潜力，但还有更多的生产用例有待观察。Sergey预计 FHE 将需要大约相同的资金才能达到目前的水平需要大约 9 亿美金 VC。这是因为 FHE 软件栈的类似部分需要重写，而且可以说需要重写更多。Sergey 认为工程师应该关注更多的垂直用例，并考虑端到端的开发人员体验。当然，考虑通用 VM 环境是有利可图的，但其应针对特定的工作负载进行优化。谁将为保护隐私的固有管理费用支付美元（工程、维护和执行成本）还有待观察。FHE 领域需要更多的资金、更多的工程师、实用的用例和简单的开发环境。

Niobium团队认为随着 2025 年的到来，FHE 是一项即将成为主流的隐私增强技术。FHE 在计算过程中对数据进行加密，这样即使被拦截也无法读取。硬件加速、风险投资和 FHETCH 等新的跨行业联盟为 FHE 带来了强劲的风向，未来两年将迎来首批商业应用。在 2025 年，随着 FHE 的正式采用，预计会观察到三个趋势：人工智能和区块链的采用将从最前沿开始；执法部门将积极推动加密技术的广泛应用；行业协作开发，而不是秘密作战。

9 参考文献

- GGH 公钥密码系统 Public-key cryptosystems from lattice reduction problems
- Regev05 论文 On Lattices, Learning with Errors, Random Linear Codes, and Cryptography

- Regev09 LWE On lattices, learning with errors, random linear codes, and cryptography
- Micciancio-Regev 书 Lattice-based cryptography
- KTX07 论文 Lattice-based cryptography
- DGHV10 论文 Fully homomorphic encryption over the integers
- LV10 论文 On Ideal Lattices and Learning with Errors over Rings
- Gentry09 Fully homomorphic encryption using ideal lattices
- Alice Silverberg Fully homomorphic encryption for mathematicians
- CR16 论文 Recovering Short Generators of Principal Ideals in Cyclotomic Rings
- BV11 论文 Efficient Fully Homomorphic Encryption from (Standard) LWE
- LTV13 论文 On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption
- GSW13 论文 Homomorphic Encryption from Learning with Errors:Conceptually-Simpler, Asymptotically-Faster, Attribute-Based
- TRLWE18 论文 TFHE: Fast Fully Homomorphic Encryption over the Torus
- 陈智罡 全同态加密：从理论到实践
- 周福才、徐剑 格理论与密码学
- Google Jeremy Kun 2024.5 博客 A High-Level Technical Overview of Fully Homomorphic Encryption
- Optalysys 团队 2024 年 10 月博客 Fully Homomorphic Encryption industry leaders join forces to form global FHE Hardware Consortium
- Fhenix 团队 2023 年 10 月 17 日博客 The Holy Grail of Encryption: The Rise of FHE Technology
- NGC Ventures 团队 2024 年 1 月 12 日博客 Introduction to FHE and Blockchain: Implications for Scalability and Privacy
- Awesome Fully Homomorphic Encryption (FHE) x Blockchain resources, libraries, projects, and more
- Awesome Homomorphic Encryption
- PANews 2024 年 9 月 17 日文章 全同态加密（FHE）的进展与应用
- 2022 年论文《Survey on Fully Homomorphic Encryption, Theory, and Applications》
- Craig Gentry 2022 年论文《Homomorphic encryption: a mathematical survey》
- 2024 年 12 月 20 日 Jorge Myszne，Niobium Microsystems 首席产品官博客 3 Homomorphic Encryption Trends for 2025
- HCLTech 团队 2024 年 6 月研报 Homomorphic encryption: Exploring technology trends and future approach
- 2024 年论文《vFHE: Verifiable Fully Homomorphic Encryption》
- PPS Lab 团队 2023 年 10 月博客 Arithmetizing FHE in Circom
- PPS Lab 团队 2023 年 10 月博客 A primer on the FHEW & TFHE schemes
- 2024 年论文《Oraqle: A Depth-Aware Secure Computation Compiler》
- 2023 年 10 月论文《A Survey on Implementations of Homomorphic Encryption Schemes》
- Sunscreen 团队 2021 年 8 月博客 An Intro to Fully Homomorphic Encryption for Engineers
- TFHE：2016 年论文《Faster Fully Homomorphic Encryption: Bootstrapping in less than 0.1 Seconds》
- BGV：2011 年论文《Fully Homomorphic Encryption without Bootstrapping》

- BFV：2012 年论文《Somewhat Practical Fully Homomorphic Encryption》
- CKKS：2016 年论文《Homomorphic Encryption for Arithmetic of Approximate Numbers》
- HomomorphicEncryption.org 2018 年 slide Building Applications with Homomorphic Encryption
- Greenfield Capital 2024 年 7 月博客 The muted Seven –7 things you should consider re confidential compute
- Sunscreen 2023 年 8 月博客 How SNARKs fall short for FHE
- 2024 年 4 月 18 日 BigBrainVC 团队博客 Flawed Homomorphic Encryption
- 2024 年 Verisense Network slide An Introduction To FHE and Its Application in Rust
- Vitalik 2020 年 7 月 20 日博客 Exploring Fully Homomorphic Encryption
- 2023 年论文《Verifiable Fully Homomorphic Encryption》
- 2018 年《Homomorphic Encryption Standard v1.1》
- 2021 年论文《SoK: Fully Homomorphic Encryption Compilers》
- 2024 年论文《SoK: Fully Homomorphic Encryption Accelerators》
- Sunscreen 团队 2023 年 5 月博客 Building an FHE compiler for the real world
- Zama 团队 2023 年 5 月 23 日博客 Private Smart Contracts Using Homomorphic Encryption
- Craig Gentry 2024 年 6 月分享视频 FHE: Past, Present and Future