第二代 FHE 关键技术原理介绍

华强胜 陈祎

大数据技术与系统国家地方联合工程研究中心,服务计算技术与系统教育部重点实验室

华中科技大学计算机学院

第二代全同态加密（FHE）方案因其在实际应用中展现出的优秀同态计算性能，使其在第三代 FHE 方案（如 GSW2013[5]）问世近十年的今天依然成为实际应用的首选。虽然没有第三代 FHE 低噪声增长和高效自举等特性，但基于环上容错学习（RLWE）问题的第二代 FHE（如 BFV、BGV 和 CKKS 方案）因为具备丰富的明文编码机制（编码浮点数、SIMD 技术）和高效地底层多项式计算框架（如余数系统（RNS）框架）等优势，在众多 FHE 方案中脱颖而出。

对于第二代 FHE 方案来说，尽管部分 SIMD 操作（如密文旋转）可能引入更多高昂计算开销的密钥切换操作,但更重要的是它能使算法执行同态操作的吞吐量提升成千上万倍，文[9]指出部分应用已利用 SIMD 将性能提升多个数量级。然而，它也是 FHE 中最复杂的技术之一，需要深入理解所涉及的 FHE 方案，进行延迟和吞吐量之间的权衡。同时由于 FHE 本身允许的同态操作较少，导致利用 SIMD 的优化方案和具体应用高度相关[1]，并且很大程度区别于明文下的算法设计。目前单机上已有的 FHE 应用也尚未形成统一算法设计范式,故设计基于 FHE 应用的高效算法需要对支持 SIMD 的打包技术（packing）和密钥切换（key switching）操作有较为深入的理解。

讨论这些基于 RLWE 问题的 FHE 方案时，始终都涉及的代数结构是多项式环 $R = \mathbb{Z}[x]/\Phi_m(x)$，其中$\Phi_m(x)$是$m$阶分圆多项式。无论明文或密文多项式，其系数都在某个模数系统下,因此实际考虑的多项式环为$R_p = R/pR = \mathbb{Z}_p[x]/\Phi_m(x)$，其中$p$是正整数并且通常称明文模为$t$，密文模为$p$（后面均用此记号）。

1. 明文结构

SV 方案将 SIMD 技术引入 FHE 中，即允许将整数环$\mathbb{Z}_t$上多个数编码为多项式环R_t上的一个多项式，使得一个多项式运算蕴含多个数的逐项运算，因此我们首先需区分“明文”概念：FHE 明文空间为R_t，这里明文指代**明文多项式**，但实际

应用中很少需要同态多项式运算，因此通常会把所编码$\mathbb{Z}_t$上的数据称为**明文数据**。当上下文中不存在歧义时，我们直接用**明文**简称。

为了应用 SIMD 技术，通常取明文模$t = p^r$（p为质数），此时明文空间为$R_{p^r} = \mathbb{Z}_{p^r}[x]/\Phi_m(x)$。我们知道当 $\gcd(m,p) = 1$ 时，$\Phi_m(x)$可以在模p意义下分解为l个不可约的d次多项式，即$\Phi_m(x) = \Pi_{i=1}^{l} F_i(x)$，其中其中$d = |p|_{\mathbb{Z}_m^*}$（$|\cdot|_G$表示$\cdot$在群$G$中的阶，$\mathbb{Z}_m^*$表示$\mathbb{Z}_m$乘法群），$l = \phi(m)/d$，$\phi(\cdot)$是欧拉函数。于是可直接得到下面环同构关系，

$$\frac{\mathbb{Z}_{p^r}[x]}{\Phi_m(x)} \cong \oplus_{i=1}^{l} \frac{\mathbb{Z}_{p^r}[x]}{F_i(x)}$$

由中国剩余定理（CRT），明文环可以下面一系列更小环的直和表示：$\alpha(x) \mapsto (\alpha(x) \bmod F_1(x), \ldots, \alpha(x) \bmod F_l(x))$，即任意$\alpha(x) \in R_{p^r}$对应$\alpha(x)$模$\Phi_m(x)$各因子后的直和。直和上的运算为逐项加和乘，称每一项为明文数据槽。可以证明$\mathbb{Z}_{p^r}[x]/F_i(x)\ (i = 1, \ldots, l)$互相同构[3]，因此明文槽的代数结构可以被唯一确定；用m次本原单位根ζ表示$x \in R_{p^r}$的剩余类，明文槽结构可表示为$E = \mathbb{Z}_{p^r}[\zeta]$（$E$中每个元素形如$a_0 + a_1\zeta + \cdots + a_{m-1}\zeta^{m-1}$），进而上述对应关系可简化为：$\alpha(x) \leftrightarrow \{\alpha(\zeta^k)\}_{k \in S}$，其中$S \subseteq \mathbb{Z}$是商群$\mathbb{Z}_m^*/\langle p \rangle$的代表元集合[6, 7]，可称每一个$\zeta^h$为对应明文槽的“赋值点”。

总体来看，第二代 FHE 组织明文槽的方式有两种：一维线性和超立方体结构。两种方式在实际参数选取和性能上皆有取舍，因此选择恰当的明文槽结构再辅以合理的打包方式设计算法是降低计算复杂度的关键。

1.1. 一维线性结构

一维明文槽结构以线性的方式理解群S的结构，下面通过具体例子说明这种线性管理方式：假设$m = 16$，由分圆多项式定义可以算得$\Phi_{16}(x) = x^8 + 1$（这也是实际应用中通常直接将$\Phi_m(x)$写作$x^n + 1$（n为 2 的幂）的原因）。取$p = 17, r = 1$，由于$d = |17|_{\mathbb{Z}_{16}^*} = 1$，故模$p = 17$ 意义下可将$\Phi_{16}(x)$分解为一次因式，即$x^8 + 1 = \Pi_{i=0}^{7}(x - \zeta^{2i+1})$。可任意取 16 次本原单位根，不妨设$\zeta = 3$（$|3|_{\mathbb{Z}_{17}} = 16$）。

如果$S=\mathbb{Z}_{16}^{*}/\langle 17\rangle\cong\mathbb{Z}_{16}^{*}$用$\{2i+1\mid i=0,\ldots,7\}$的方式生成，则明文空间中任意多项式$f$可以看作如图 1 所示编码 8 个数的明文槽结构，并通过下标$i(=0,\ldots,7)$索引。譬如按顺序编码(3,16,15,2,3,16,15,2)，经过多项式插值可以算得编码它们

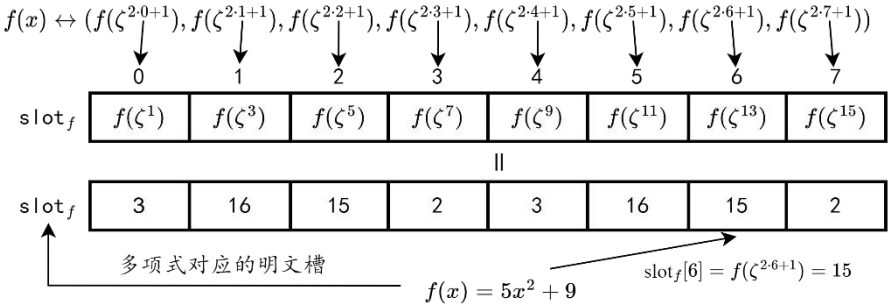
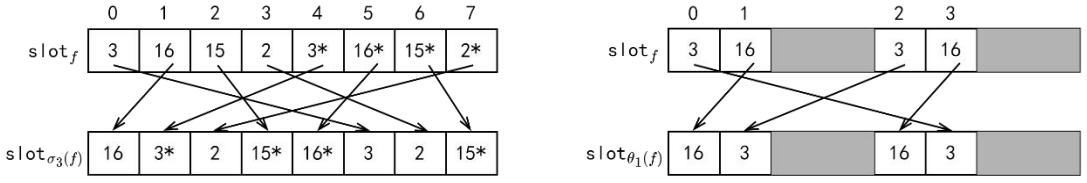


图 1：一维明文槽结构与编码解码关系

的多项式$f(x)=5x^2+9$，获取 6 号槽的数据需通过计算f在对应赋值点处的值$\mathrm{slot}_f[6]=f(\zeta^{2\cdot6+1})=15$得到。需要注意的是，$\Phi_{16}(x)$已完全分解，故$e=8$是该参数设定下的最大编码量。

该编码技术除了能够使多项式的一次加法和乘法运算蕴含$\mathbb{Z}_{p^r}$上多个数的逐项运算，还能通过R_{p^r}上的自同构（automorphism，即$R_{p^r}\to R_{p^r}$上的双射，通常用σ表示）实现槽内元素同态地置换，也即第二代 FHE 中密文旋转（rotation）操作。这两个特性构成第二代 FHE 方案 SIMD 技术的完整内涵。实际上，并非所有线性管理方式都能找到实现**循环移位**的自同构。譬如按照上面参数设定，取自同构$\sigma_j:x\mapsto x^j$，只能实现$\{[((2i+1)\cdot j \bmod 16)-1]/2\}$号槽到$i$号槽这样相对散乱的置换；取$j=3$，对图 1 中多项式$f$使用$\sigma_3$其对应槽内发生的置换如图 2(a)所示（由于$[((2\cdot5+1)\cdot3 \bmod 16)-1]/2=0$，故原来的 0 号槽内数据移至 5 号槽内，图中“*”起区别作用）。



(b) 赋值点选取方式一　　(b) 赋值点选取方式二

图 2：自同构对应的置换：图(b)实现循环移位

为实现循环移位效果的密文旋转，需要将赋值点规定为$\zeta^{g^i}(i=0,1,\ldots)$的形式，通常称$g$为生成元。在上面参数设定下$S$中元素的最大阶为 4，例如$3\in S$生成$T=\{3^1=3,3^2=9,3^3=11,3^4=1\}$（模 16 意义下）。取$g=3$，用$i=0,1,2,3$来索引明文槽(赋值点仅使用$T\subseteq S$中的元素作上标)，于是定义自同构$\theta_j=\sigma_{3^j}:x\mapsto x^{3^j}$

可以实现明文多项式对应的槽位循环左移j位。这是因为$\theta_j(f)=f\left(x^{3^j}\right)$，此时$\theta_j(f)$对应的$i$号槽位值为$\left(\theta_j(f)\right)\left(\zeta^{3^i}\right)=f\left(\zeta^{3^{i+j}}\right)$，如图 2(b)中取$j=1$实现循环左移 1 位操作。然而，图 2(b)中灰色槽相当于将$k\in S-T$作为上标的ζ^k视作无效赋值点，故编码量仅为$e=4$，编码率为 0.5。

1.2. 超立方体结构

一维明文槽结构为实现密文旋转不得不浪费一些槽位。但如果将明文槽看作更高维结构，即用超立方体（hypercube）结构管理时，不仅可以增大明文数据编码率，还可以实现某一维度上对超立方体内所有超列（hypercolumn）的循环移位，以支持更灵活的应用。因此，与其只用一个生成元生成S的子集，不如考虑用多个生成元完整生成$S=\left\{g_1^{e_1}\cdots g_t^{e_t}\mid 0\le e_i<D_i,i=1,\ldots,t\right\}$，其中$\{g_i\mid(i=1,\ldots,t)\}$组成 t 维超立方体基（hypercube basis）。保持第 1.1 小节参数设定，一种可行超立方体基的选取方式为$t=2,g_1=3,g_2=7,D_1=4,D_2=2$，此时恰好使得$\{3^{e_1}\cdot 7^{e_2}\mid 0\le e_i<D_i,i=1,2\}$能不重不漏地生成$S$（$1=3^0\cdot 7^0,3=3^1\cdot 7^0,5=3^1\cdot 7^1,7=3^0\cdot 7^1,9=3^2\cdot 7^0,11=3^3\cdot 7^0,13=3^3\cdot 7^1,15=3^2\cdot 7^1$，满

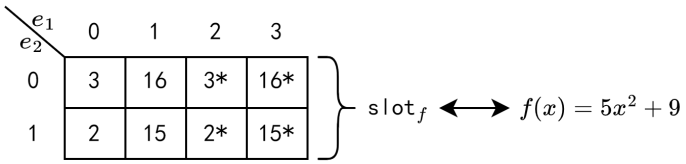


图 3：二维明文槽结构

足$|S|=D_1\cdot D_2$）。令$k=3^{e_1}\cdot 7^{e_2}$，赋值点为ζ^k，于是可用二维坐标(e_1,e_2)索引如图 3 所示的二维明文槽结构，可看出编码的明文数据和对应的多项式均未发生改变，仅仅只是改变了组织方式，其中$\mathrm{slot}_f[e_1=1][e_2=1]=f(3^{3^1\cdot 7^1})=15$。

作为一维形式的拓展，使用超立方体结构组织方式时，密文旋转对明文槽的作用规定为“rotation1D”，记作ρ_s^j以表示s-维上所有超列循环左移j位。例如对上述二维明文槽，密文旋转可以实现行或列之间的循环移位（1-维表示列、2-维表示行）。ρ_s^j最多可以通过两个自同构实现：$\rho_s^j=\mu\sigma_u+(1-\mu)\sigma_v$，其中$u=g_s^{-j}(\mathrm{mod}\,m),v=g_s^{D_s-j}(\mathrm{mod}\,m)$；掩码多项式$\mu$对应明文槽是按特定要求编码数据的超立方体$((e_1,\ldots e_s,\ldots,e_t),e_s<j$索引的明文槽内数据为 0，其余为 1)；在 HElib 库中，若有$D_s=|g_s|_{\mathbb{Z}_m^*}$，则称$s$-维为“好”维度，否则为“坏”[4]。当$s$-维为“好”

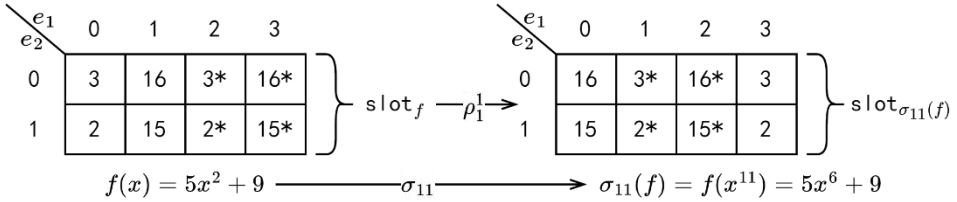


图 4：rotation1D(1,1)示意图

时，仅需一个自同构σ_u即可实现ρ_s^j。譬如上述参数设定下，由于$|3|_{\mathbb{Z}_{16}^*} = D_1, |7|_{\mathbb{Z}_{16}^*} = D_2$，故 1-维和 2-维均为“好”维度，于是$\rho_1^1 = \sigma_{11}: x \mapsto x^{11}$（$g_1^{-1} = 3^{-1} \bmod 16 = 11$）可以实现如图 4 所示的列循环左移 1 位。

2. 密钥切换

基于 RNS 框架的 FHE 实现中，由于同态乘和密文旋转操作都包含密钥切换，导致这些同态操作在一个渐进计算复杂度水平上。可见密钥切换主导同态操作计算开销，理解它出现的原因对设计基于 FHE 的低复杂算法有重大意义。密钥切换源自 BV 方案[2]中处理密文维度增长问题的重线性化技术，但并不能简单在两者之间画等号。只有结合第二代 FHE 的密文旋转操作，才能赋予密钥切换完整内涵：将被同态运算潜在影响的私钥复原，使得明文数据持有者能用原私钥正确解密。

下面根据使用密钥切换的两种场景简单说明其产生原因，最后综合讨论它们对密钥切换矩阵（key-switching matrix，也被称作旋转密钥(rotation key)）产生的影响，以及分布式中如何简单避免目前单机上存在的和密钥切换矩阵有关问题，并提出分布式场景如何给第二代 FHE 上层应用提供进一步降低密钥切换次数的可能性。

2.1. 同态乘引起密钥切换

以 BGV 方案为例，首先引入新记号$R_q[y]$表示变量为y、各系数属于R_q的多项式。于是密文 $\mathrm{ct} = (c_1, c_2) \in R_q^2$可以看成$R_q[y]$中元素，写作 $\mathrm{ct}(y) = c_1 + c_2 \cdot y$。若 ct 对应的解密私钥为 $\mathrm{sk} = (1, s)$，则 BGV 解密可写作$(\mathrm{ct}(s) \bmod q) \bmod t$。

同态乘法接受两个密文 $\mathrm{ct} = (c_1, c_2), \mathrm{ct}' = (c_1', c_2')$，按下述两步处理：首先是进行$R_q[y]$上的乘法（$\mathrm{ct}^{\times}(y) = \mathrm{ct}(y) \cdot \mathrm{ct}'(y)$，也被称作原始乘法（raw multiplication)），算得维数增长的密文$\mathrm{ct}^{\times} = (c_1 c_1', c_1 c_2' + c_2 c_1', c_2 c_2')$。为降低$\mathrm{ct}^{\times}$的维数，第二代 FHE 会在密钥生成阶段生成密钥切换矩阵（作为公钥的一部分），给随后的密钥切换提供足够的辅助信息以将$\mathrm{ct}^{\times}$转换为$\mathrm{ct}'' = (c_1'', c_2'')$。从另一个视角来看，第一步计算由输入密文算得结果$\mathrm{ct}^{\times}$，对应解密私钥由 $\mathrm{sk} = (1, s)$ 转变为 $\mathrm{sk}^{\times} = (1, s, s^2)$（$\mathrm{ct}^{\times}(s) = c_1 c_1' + \left(c_1 c_2' + c_2 c_1'\right) \cdot s + c_2 c_2' \cdot s^2 = (c_1 + c_2 s)\left(c_1' + c_2' s\right) = \mathrm{ct}(s) \cdot \mathrm{ct}'(s)$），为了保证私钥持有者能够正确解密，同态乘法结果$\mathrm{ct}''$必须要能被 sk 解密。因此总体来看，密钥切换不仅使密文维度降低，同时还将同态乘法对私钥产生的影响消除。

2.2. 密文旋转引起密钥切换

$m \in R_t = \mathbb{Z}_t[x]/\Phi_m(x)$是关于变量$x$的多项式，故可写作$m(x)$。考虑明文$m(x)$的加密 $\mathrm{ct}(x) = (c_1(x), c_2(x)) \in R_q^2$（类似地，$\mathrm{ct}(x)$表示该向量各分量是关于$x$的多项式），其对应的解密私钥是 $\mathrm{sk}(x) = \big(1, s(x)\big)$。由于密码系统的正确性，它们之间满足 $\langle \mathrm{ct}(x), \mathrm{sk}(x)\rangle = m(x) + t \cdot e(x) \bmod q$，于是 $\mathrm{ct}(x)$ 的解密过程为 $(\langle \mathrm{ct}(x), \mathrm{sk}(x)\rangle \bmod q) \bmod t$。然而，如果对 $\mathrm{ct}(x)$应用自同构$\sigma_i: x \mapsto x^i$，解密 $\mathrm{ct}(x^i)$的私钥将变为 $\mathrm{sk}(x^i)$。也就是说 $\mathrm{sk}(x)$无法解密原始密文旋转（raw rotation）后的密文 $\mathrm{ct}(x^i)$，应该对其进行密钥切换以生成能够被 $\mathrm{sk}(x)$解密的密文，即让原始密文旋转对私钥产生的影响消除。

2.3. 密钥切换矩阵

密钥切换所需输入除待处理密文外，还有密钥切换矩阵。根据上面分析，密钥切换使用场景相对固定，基本可以确定私钥的始末状态。实际上，密钥切换矩阵是将私钥变化数据加密后公开，而密钥切换的作用正是将这些辅助信息加入待处理密文以实现初始私钥可正确解密。由于辅助信息可能引入较大的噪声，一个很自然的、也是目前广泛采用的策略是利用基数分解减缓噪声增长。然而，这一方面使得密钥切换计算开销极大增加，另一方面又使密钥切换矩阵所占存储空间增大，给单机上 FHE 应用带来很多问题。

考虑到存储问题，目前单机上实现同态乘操作时，处理原始同态乘的密钥切换矩阵（简称乘法密钥切换矩阵）只有一种，记作$W[\mathrm{sk}^{\times} \to \mathrm{sk}]$，其中记号 sk、$\mathrm{sk}^{\times}$含义同上；处理原始密文旋转的密钥切换矩阵（简称旋转密钥切换矩阵）数量与算法中所需的自同构σ_i总数有关（最多可能有$\phi(m)$个），分别记作$W[\sigma_i(\mathrm{sk}) \to \mathrm{sk}]$。因此旋转密钥切换矩阵数量对算法存储复杂度有决定性影响。文[7]称即使只考虑处理超立方体明文槽内 rotation1D 和 Forbenius 映射对应的密钥切换矩阵，将它们完全存储也需要几百 GB 内存。这引导我们设计密文旋转种类更少的算法，例如文[7]采用大步小步的存储策略正是为了适配基于大步小步的矩阵-向量乘法。而分布式场景中，通过合理的数据划分让无关计算于每轮中并行执行，因此允许将每台处理器计算所需的密文旋转种类降至尽可能低，从而降低密钥切换矩阵数量。

分布式内存模型下，各台处理器所需密钥切换矩阵数量的降低可能带来了一

些额外存储空间，允许它们存放一些额外的乘法密钥切换矩阵，以便实现密钥切换**延后**优化，进一步减少密钥切换次数从而降低计算复杂度。

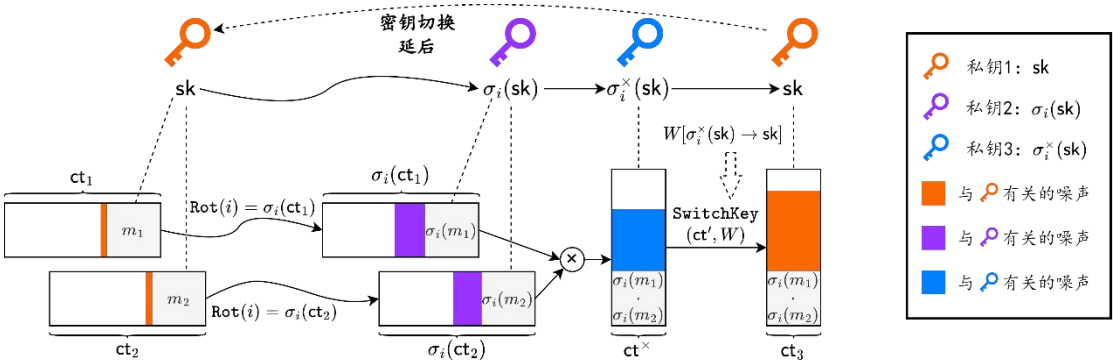


图 5 密钥切换延迟优化

图 5 描述这一优化的大致过程，其中密文被形象地看作是用噪声掩盖的明文（安全性由 RLWE 假设保障），用相同颜色标记私钥和噪声表明私钥只能解密相应的密文。图中密文$\mathrm{ct}_1, \mathrm{ct}_2$依次同态地进行旋转，但每次只做原始密文旋转，并不会立即执行密钥切换；当两密文需要做同态乘法时，若它们对应明文槽数据相比初始状态的位移量一致，换句话说，此时它们对应的私钥状态相同，可以直接执行$R_q[y]$上的原始乘法。随后只需提供新的乘法密钥切换矩阵$W[\sigma_i^{\times}(\mathrm{sk}) \to \mathrm{sk}]$即可将结果密文对应私钥恢复至初始状态。需注意单机场景中，一方面应用所需密文旋转种类一般都比较多，也即旋转密钥切换矩阵数目本身所占存储空间较大；另一方面，如果还像上述一样考虑经原始旋转的密文之间的同态乘，势必造成私钥更多种状态改变，进而引入大量乘法密钥切换矩阵，故单机很容易因存储空间受限导致难以实现密钥切换延后优化。

参考文献

[1] Blatt M, Gusev A, Polyakov Y, et al. Secure large-scale genome-wide association studies using homomorphic encryption. Proceedings of the National Academy of Sciences, 2020, 117(21): 11608-11613.

[2] Brakerski Z, Vaikuntanathan V. Efficient fully homomorphic encryption from (standard) LWE[J]. SIAM Journal on computing, 2014, 43(2): 831-871.

[3] Gentry C, Halevi S, Smart N P. Better bootstrapping in fully homomorphic encryption. PKC 2012:1-16.

[4] Gentry C, Halevi S, Smart N P. Fully homomorphic encryption with polylog overhead. Eurocrypt 2012: 465-482.

[5] Gentry C, Sahai A, Waters B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. CRYPTO (1) 2013: 75-92.

[6] Halevi S, Shoup V. Design and implementation of HElib: a homomorphic encryption library. IACR Cryptol. ePrint Arch. 2020, 1481:1-42.

[7] Halevi S, Shoup V. Faster homomorphic linear transformations in HElib. CRYPTO 2018: 93-120.

[8] Smart N P, Vercauteren F. Fully homomorphic encryption with relatively small key and

ciphertext sizes. PKC 2010: 420-443.
[9] Viand A, Jattke P, Hithnawi A. SoK: Fully homomorphic encryption compilers. S&P 2021: 1092-1108.