计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(7): 1738−1753, 2025

基于容错学习问题的全同态加密算法和硬件优化综述

河人华[1] 李 冰[2] 杜一博[3,4] 王 颖[3] 李晓维[3] 韩银和[3]

[1](首都师范大学数学科学学院 北京 100048)

[2](北京国家应用数学中心(首都师范大学) 北京 100048)

[3](中国科学院计算技术研究所 北京 100190)

[4](中国科学院大学 北京 101408)

(2220502119@cnu.edu.cn)

Survey on Algorithm and Hardware Optimization to LWE-Based Fully Homomorphic Encryption

He Renhua[1], Li Bing[2], Du Yibo[3,4], Wang Ying[3], Li Xiaowei[3], and Han Yinhe[3]

[1](*School of Mathematical Sciences*, *Capital Normal University* , *Beijing* 100048)

[2](*Beijing National Center for Applied Mathematics* (*Capital Normal University*), *Beijing* 100048)

[3](*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)

[4](*University of Chinese Academy of Sciences*, *Beijing* 101408)

Abstract With the rapid development of cloud computing, quantum computing and other advanced technologies, data privacy is facing increasingly severe threats. Especially in recent years, more and more users have been storing their sensitive data and applications in the cloud to take advantage of convenient services and powerful computing capabilities. However, traditional security technologies can not fully guarantee the security of cloud computing. Introducing fully homomorphic encryption algorithms is one of the effective ways to address this issue. At the same time, fully homomorphic encryption technology based on lattice theory has the capabilities of natural resistance to quantum attacks and arbitrary calculations on data in an encrypted state, effectively guaranteeing data security in the quantum computing era. Although fully homomorphic encryption shows significant potential, it suffers the problem of the volume explosion of computing and storage. To address the above problem and speed up the widespread adoption of fully homomorphic encryption algorithms, researchers from the fields of algorithms and hardware have proposed a variety of solutions, and significant progress has been made. This work summarizes the progress of mainstream fully homomorphic encryption technology, analysis and compilation of algorithm libraries and fully homomorphic hardware accelerator in the past five years, and finally provides perspective of fully homomorphic encryption technology in the future.

Key words fully homomorphic encryption; homomorphic encryption algorithm; homomorphic encryption library; fully homomorphic encryption hardware accelerator; learning with errors

摘 要 随着云计算、量子计算等技术的飞速发展,数据隐私面临严峻威胁.越来越多的用户将数据和应用程序存储在云端,但传统的安全技术难以保障云计算环境中的数据安全.在此背景下,引入全同态加密算

收稿日期: 2023-12-14; **修回日期**: 2025-03-05

基金项目: 国家自然科学基金项目(62204164); 北京市教育委员会项目(KM202310028001)

This work was supported by the National Natural Science Foundation of China (62204164) and the Project of Beijing Education Committee (KM202310028001).

通信作者: 李冰(bing.li@cnu.edu.cn)

法成为有效的解决方案之一. 同时，基于格理论的全同态加密技术具有天然的抗量子攻击能力，能够在加密状态下对数据进行任意计算，有效地为量子计算时代数据安全提供保障. 尽管全同态加密有广阔的应用前景，但它存在计算和存储巨额开销的问题. 为了推动全同态加密算法的应用和落地，算法和硬件领域的研究人员提出了多种解决方案并取得显著进展. 归纳了主流的全同态加密技术以及分析整理算法计算库和全同态硬件加速的近 5 年相关工作的进展，最后展望了全同态加密技术.

关键词 全同态加密；同态加密算法；全同态加密算法库；全同态加密硬件加速器；容错学习

中图法分类号 TP309.7；TP303

DOI: 10.7544/issn1000-1239.202331022 **CSTR:** 32373.14.issn1000-1239.202331022

随着云服务、人工智能等技术的飞速发展，数据安全面临严峻挑战，世界各地频发数据泄露和黑客攻击事件，损失难以估量[1]. 同时，量子计算机的快速发展对现代公钥加密方案构成了严重威胁. 现代公钥加密方案的安全性主要依赖于整数分解和离散算法等数学难题的复杂性，然而量子计算机可能有能力在有限时间内破解上述问题. 全同态加密（fully homomorphic encryption, FHE）是一种基于格密码的加密方案，具有可证明的对抗量子攻击的安全性，允许在密文不被解密的条件下执行多个对密文的操作[2]. 因此，全同态加密已成为后量子密码标准化进程中的核心候选方案之一，并获得了来自学术界和工业界的广泛关注.

全同态加密是一种允许在加密状态下进行任意加法和乘法组合运算的加密方案. 在 1978 年公钥加密体制被提出时，Rivest 等人[3] 就发现了 RSA 方案具有部分同态性，并在云计算和金融交易等场景应用. 但 RSA 方案的安全性基于 2 个大素数乘积难解的困难假设，存在一定的安全风险. 2009 年，Gentry[2] 提出了基于理想格的全同态加密方案，此工作为后续的全同态加密方案提供了设计蓝图. 初期的全同态加密方案由于昂贵的计算开销限制了其实际应用. 经过近十几年的不懈努力，研究人员改进了全同态加密的性能，其中代表性的方案有 BGV[4], BFV[5-6], GSW[7], FHEW[8], TFHE[9], CKKS[10]. 这些方案基于不同的安全假设，将全同态计算的数据从整数域拓展到实数域，并持续降低了全同态加密中自举（bootstrapping）运算的计算开销. 目前，全同态加密方案在医疗诊断、云计算、生物医药、机器学习等领域展示了广泛的应用前景[11-14].

全同态加密算法在取得快速发展的同时，其计算效率仍面临挑战. 由于全同态加密算法是计算密集型应用，其参数数值庞大并且数据规模巨大，因此全同态加密硬件加速已成为当前研究的热点，主要集中在传统处理器优化和专用硬件加速器设计两大方向上. 硬件加速面临的关键挑战在于能有效地利用数据的并行处理能力，并在有限的内存带宽下合理地管理数据流. 目前的硬件加速研究不仅提出了适应不同硬件平台特性的算法优化方法，还探索了旨在提高数据重用率和硬件利用率的架构设计方案.

尽管全同态加密领域的基础算法、算法库与硬件加速优化近年来发展迅速，并已有综述文章对此进行了总结[15-17]，但目前尚缺乏将这几方面工作综合调研的工作. 本文旨在探讨它们之间协同优化的研究成果，以支持全同态加密领域的进一步发展. 本文梳理了近 5 年全同态加密领域的相关工作，涵盖全同态加密算法、算法库及硬件加速平台，分析了各项工作在解决全同态加密算法部署中的主要挑战时所提出的解决方案. 此外，本文总结了全同态加密领域的发展趋势，并对未来全同态加密研究的发展方向进行了展望.

1 背　　景

1.1 基于 LWE 问题的全同态加密概述

与传统公钥加密方案不同，全同态加密可以在不解密的情况下对密文进行有意义计算，保护数据在传输、存储和使用时的安全性. 目前流行的全同态加密方案的构建主要基于格上的困难问题：容错学习（learning with errors, LWE）问题[18] 及其变体环容错学习（ring learning with errors, RLWE）问题[19]，相比于早期全同态加密方案，如今全同态加密的计算和困难假设都更容易实现. 基于 LWE 问题和 RLWE 问题的加密方案的密码的构建流程类似，为了保证方案安全性引入了噪声. LWE 问题和 RLWB 问题方案之间的主要区别在于，明文、密文和密钥所在的数域和计算效率的差异. 本文所用符号如表 1 所示.

全同态加密方案ε是由 1 组概率多项式时间

Table 1 Symbol Explanation

表 1 符号说明

符号	含义
$KeyGen_{\varepsilon}$	加密方案ε的密钥生成算法
Enc_{ε}	加密方案ε的加密算法
Dec_{ε}	加密方案ε的解密算法
$Eval_{\varepsilon}$	加密方案ε的密文计算算法
λ	安全参数
pk	公钥
sk	私钥
$\boldsymbol{P}$	公钥为矩阵时用$\boldsymbol{P}$表示
$\boldsymbol{s}$	私钥为向量时用$\boldsymbol{s}$表示
evk	评估密钥
$\boldsymbol{c}=(c_1,c_2,\cdots,c_t)$	密文
f	密文函数
$\boldsymbol{c}_f$	密文计算后获得的密文
$\boldsymbol{m}=(m_1,m_2,\cdots,m_t)$	明文
u	当明文为单比特明文时写作u
e	噪声
$q\in\mathbb{Z}$	模数q
$\mathbb{Z}_q$	以q为模的整数集合
$\mathbb{Z}_q^n$	$\mathbb{Z}_q$上的n维向量
$R=\mathbb{Z}[x]/\langle f(x)\rangle$	多项式环, 其整数系数以（单次）多项式$f(x)$为模
$R_q=\mathbb{Z}_q[x]/\langle g(x)\rangle$	多项式的系数是模q的多项式环

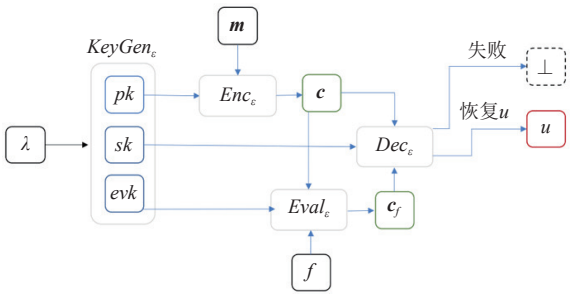


Fig. 1 Fully homomorphic encryption scheme

图 1 全同态加密方案

(probabilistic polynomial-time)算法组成, 图 1 表述了全同态加密方案的构建流程.

1) $KeyGen_{\varepsilon}(\lambda)$. 根据安全参数$\lambda$, 生成密钥$sk$、公钥$pk$和评估密钥$evk$, 基于(R)LWE 格上的困难问题, 通过随机选择误差项构造以q为模的公钥和私钥. evk会在密文计算中用到, evk的形式在不同全同态方案中不同.

2) $Enc_{\varepsilon}(pk,\boldsymbol{m})$. 根据公钥$pk$, 对明文消息$\boldsymbol{m}$进行加密, 加密中引入噪声$e$, 产生密文$\boldsymbol{c}$. 密文向量$\boldsymbol{c}$基于(R)LWE 问题生成, 以$q$为模.

3) $Dec_{\varepsilon}(sk,\boldsymbol{c})$. 根据密钥$sk$和密文$\boldsymbol{c}$, 输出明文消息$\boldsymbol{m}$. 计算密文和密钥向量的内积(即在密钥下计算多项式方程的解), 最后通过模q恢复明文. 如果解密算法无法成功恢复加密消息$\boldsymbol{m}$(一般是噪声e溢出的情况), 则该算法会提供解密失败输出$\perp$.

4) $Eval_{\varepsilon}(evk,(c_1,c_2,\cdots,c_t),f)$. 同态评估计算. 输入评估密钥$evk$、密文函数$f$、密文$(c_1,c_2,\cdots,c_t)$, 输出一个密文$\boldsymbol{c}_f$, 其中密文函数$f$定义了一个计算任务. 密文$\boldsymbol{c}_f$解密后得到$f(m_1,m_2,\cdots,m_t)$. 即$Dec_{sk}(\boldsymbol{c}_f)=f(m_1,m_2,\cdots,m_t)$. 密文$\boldsymbol{c}_f$和密文$\boldsymbol{c}$实质上是基于相同的明文构建得到的, 但具有不同的噪声.

在基于 LWE 构建的 FHE 方案中, 明文$\boldsymbol{m}$通过引入噪声e的方式生成密文$\boldsymbol{c}$, 密文每进行一次同态操作, 噪声都会增加, 经过一定次数的乘法或加法后, 由于噪声持续增长, 将最终导致噪声溢出(超过$q/4$), 无法正确解密. 因此如何处理噪声并提高 FHE 的计算能力是构建 FHE 方案的关键问题.

早期 FHE 方案首先构造一个部分同态加密(somewhat homomorphic encryption, SHE)方案, 即能够进行有限次同态计算的方案, 再使用自举技术降低密文结果中的噪声, 得到 FHE 方案(即在密文上执行任意计算). 但 SHE 方案需时刻关注噪声大小以保证其正确解密. 因而后续更为常见的构建方案是先构造一个有限级同态加密方案(leveled fully homomorphic scheme, LHE), 再进行自举操作. LHE 方案是一种支持特定深度电路评估的方案, 保证在进行L次同态乘法时仍能正常解密, 再进行自举操作.

在 FHE 方案中, 电路深度L会影响 FHE 方案的安全性和计算效率. 选择一个大的L能够降低自举操作次数, 但需要更大的密码参数(比如模数q), 导致计算参数规模膨胀过大. 但小的L需要更频繁的自举操作及自举带来的复杂计算. 自举过程如图 2 所示, 它是密文$\boldsymbol{c}$在新密钥下的重新加密, 即在公钥pk_2下加密私钥sk_1获得新密钥$\overline{sk}_1$, 再使用新密钥同态解密新密文得到在公钥pk_2下加密的密文$\boldsymbol{c}'$. 密文$\boldsymbol{c}'$与密文$\boldsymbol{c}$有相同明文, 但具有更小噪声. 自举技术通过计

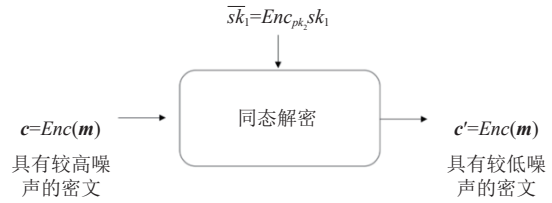


Fig. 2 Bootstrapping process in fully homomorphic encryption

图 2 全同态加密中的自举过程

算密文 $\boldsymbol{c}$ 和新密钥 $\overline{sk}_1$ 的解密函数（$Dec_{\varepsilon}(\boldsymbol{c},\overline{sk}_1)$）来刷新密文噪声，从而实现 FHE 方案.

1.2 基于 LWE 问题的全同态加密方案的发展

Brakerski 等人[20]提出了基于 LWE 问题构建的全同态加密方案，设计了全同态构建方法，并提出了重线性方法和原始的模数变换方法以实现密文计算. 因此，此工作奠定了整数域上 FHE 方案的基础.

首先构建安全性基于 LWE 假设的加密方案，使用密钥 $\boldsymbol{s}\in\mathbb{Z}_q^n$ 加密比特 $u\in\{0,1\}$，经过加密计算得到密文 $\boldsymbol{c}$. 加密计算公式为：

$$\boldsymbol{c}=Enc_{\varepsilon}(\boldsymbol{s},u)=(\boldsymbol{a},b=\langle\boldsymbol{a},\boldsymbol{s}\rangle+2e+u),$$

其中 $\boldsymbol{a}$ 为随机生成向量且 $\boldsymbol{a}\in\mathbb{Z}_q^n$，一个噪声 e，密文 $\boldsymbol{c}\in\mathbb{Z}_q^n\times\mathbb{Z}_q$. 解密关键是处理 2 个不会互相干扰的“掩码”：$\langle\boldsymbol{a},\boldsymbol{s}\rangle$ 和 $2e$. 解密算法首先重新计算 $\langle\boldsymbol{a},\boldsymbol{s}\rangle$ 的值，再计算 $b-\langle\boldsymbol{a},\boldsymbol{s}\rangle$，得到 $2e+u(\mathrm{mod}\ q)$. 需保证 e 足够小，使得 $2e+u(\mathrm{mod}\ q)=2e+u$，最后计算 $2e+u(\mathrm{mod}\ 2)$，得到 u，即解密成功.

在同态乘法的过程中，多项式形式的密文维数将从 $n+1$ 增加到约 $n^2/2$，而密文维数快速膨胀将增加计算成本，从而极大影响加密算法的性能. 为了解决密文维数膨胀问题，Brakerski 等人[20]提出了重线性化方法，将密钥 $\boldsymbol{s}$ 更换为维数更低的密钥 $\boldsymbol{t}$，将密文维数减少到 $n+1$. 此外，该工作采用模数变换（modulus switching）技术，通过将密文 $\boldsymbol{c}\in\mathbb{Z}_q$ 乘以 p/q，其中 p 充分小于 q，并取最接近的整数，将密文 $\boldsymbol{c}$ 转化成密文 $\boldsymbol{c}'\in\mathbb{Z}_p$，降低密文模数，从而实现了将 SHE 方案转化为 FHE 方案. 在此之前，Gentry[2]早期方案在稀疏子集合和问题（sparse subset sum problem, SSSP）下进行“挤压”操作来降低解密复杂度. 模数变换技术避免了引入额外假设，不仅大大降低了解密复杂度，还降低了密文中的噪声，因而在后续其他 FHE 方案中广泛应用.

虽然 FHE 方案效率有大幅提升，但重线性化方法需要将长密文向量乘以大小为 $\Omega(n^3)$ 的重线性化矩阵，计算成本较高. 针对上述问题，Gentry 等人[7]构建了引入近似特征向量的 GSW 方案. 该方案的密文加法与乘法均为自然的矩阵加法与乘法，避免密文进行同态乘法时密文维数上升，降低了计算复杂度. 同时通过使用密文展平（flattening）技术，避免了同态运算后噪声过大导致解密失败的问题，既保证了密文强有界，同时也保证了 GSW 方案可以评估深度为 L 的电路，实现 LHE 方案. 但是 GSW 方案一旦使用自举技术实现 FHE 方案，评估者需要获取新密钥 $\bar{\boldsymbol{s}}_1$（$n+1$ 维向量），即在公钥 $\boldsymbol{P}_2$（$n+1$ 阶矩阵）下加密密钥 $\boldsymbol{s}_1$（$n+1$ 维向量），相比于之前的方案，GSW 方案中公钥尺寸较大，使得自举更加昂贵. 但 GSW 方案的构造方法给后续全同态加密方案提供了重要参考和启发. 在 GSW 案基础上，FHEW[8]，TFHE[9]进一步优化自举技术，实现了快速自举.

Cheon 等人[10]在 2017 年提出了一种实数域 FHE 方案（称作 CKKS 方案），即一种支持复数域上近似计算的 LHE 方案，以满足深度学习等应用对实数域同态计算的需求. CKKS 将噪声视为近似计算期间发生错误的一部分，同时通过在加密前将明文乘以比例因子的方式，减少加密噪声造成的精度损失. CKKS 方案与文献 [20] 中的密文形式类似，通常是 1 对多项式，在没有舍入的情况下，电路深度 L 越大，密文模数越大，解密复杂度增大. 为了支持较大的电路深度，Cheon 等人[10]提出重缩放（rescaling）技术，降低了密文模数的大小，并在几乎保持明文精度的同时，减少了消息中的误差. 尽管与模数变换类似，都是对模数 q 进行优化，但重缩放技术与模数变换技术解决的问题不同. 2018 年，Cheon 等人[21]将此 LHE 方案扩展为 FHE 方案.

图 3 总结了近年来 FHE 方案的发展，通常以 BGV 和 BFV 为代表的全同态方案被归类为第 2 代全同态加密方案，而 GSW，FHEW，TFHE 代表第 3 代，以 CKKS 为典型的实数域全同态加密方案则为第 4 代. 这些不同的方案在效率和适用应用场景上有显著差异. 比如 BGV，BFV，CKKS 适合处理同态评估乘法深度较小的电路，而对于深度较大且未知的电路，FHEW 与 TFHE 方案则展现出明显的效率优势. 不同 FHE 方案的算法和计算类型存在一定的相似性，因此有工作提出同时支持多种 FHE 方案. 比如可根据计算任务特点选择 FHE 方案的工作 Chimera[22]；实现了多种方案的算法库 OpenFHE[23]；支持多种全同态加密方

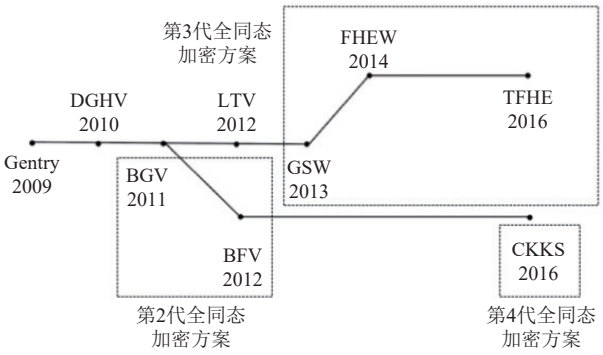


Fig. 3 Development process of typical fully homomorphic encryption scheme

图 3 典型的全同态加密方案发展过程

案的硬件加速器, 如 F1[24]. 这些进展不仅丰富了全同态加密的技术选择, 还为不同应用场景提供了定制化的解决方案, 推动了全同态加密在实际生活中的应用和发展.

2 基于 LWE 问题的全同态加密

支持算术电路和布尔电路的同态加密方案在构建基础和方案特点上有显著的不同. 在本节, 我们将具体分别介绍它们计算的特点, 并在表 2 中进行了汇总, 同时在表 3 中对部分 FHE 方案中的自举操作特点做了简单总结.

Table 2 Characteristics of Different Fully Homomorphic Encryption Schemes

表 2 不同全同态加密方案的特点

方案	共同特点	重要突破与特点
BV[20]	以算术电路为主, 用于控制密文维数增加的技术	模数变换, 重线性化技术
BGV[4]		免自举的 LHE 方案, 带有模数变换的同态乘法
BFV[5-6]		比例不变性技术以减缓噪声增长速度
RNS FV[25]		提出 FV 的 RNS 变体
HPS[26]		提出了有效的缩放和 CRT 基础扩展程序, 为 RNS FV 的优化改进
CKKS[10]		将加密噪声视为近似计算误差, 重缩放技术以减低噪声
文献 [21]		CKKS 的 FHE 方案, 利用缩放正弦函数近似和加速模归约运算
文献 [27]		CKKS 的 Full-RNS 变体, 快速基转化
文献 [28]		减少了密钥变换中临时模数的数量
GSW[7]	快速自举, 以布尔电路为主, 密文是自然相加和相乘的矩阵, 避免了密文维数膨胀的问题	提出近似特征向量方法, 提出了新的构造同态乘法的方法
文献 [29]		构建了第一个加密矩阵并支持同态矩阵运算的 FHE 方案
FHEW[8]		使用累加器做自举
TFHE[9]		提出了非对称输入的同态乘法
文献 [30]		提出了新的打包方法和电路自举方法, 无填充可编程自举
文献 [31]		提出可编程自举的多输出版本

2.1 在算术电路上实现的同态加密方案

研究学者们提出了不少高效的算术电路 FHE 方案, 如表 2 所示.

针对文献 [20] 中频繁自举操作开销过大问题, Brakerski 等人[4] 在 BGV 方案中实现了评估 L 级电路的能力, 通过反复应用模数变换以保持 L 级电路评估时噪声规模基本不变, 在此基础上, 通过自举技术将所提 LHE 方案转化为 FHE. 此外, BGV 方案采用 SIMD 将多个明文打包到每个密文, 批量处理多个密文的自举和解密, 大大提高了效率. 同时 Brakerski 等人[20] 提出的重线性化方法不仅具备降低密文维数功能, 还可以用于将可解密的密文 $\boldsymbol{c}_1$ 转换为另一个密文 $\boldsymbol{c}_2$, 但 $\boldsymbol{c}_2$ 的维度未必比 $\boldsymbol{c}_1$ 低, 因此 BGV[4] 将重线性化过程重命名为密钥变换(key switching).

Table 3 Characteristics of Bootstrapping Process in Fully Homomorphic Encryption Schemes

表 3 全同态加密方案自举过程特点

方案	是否支持多个明文同时自举	自举过程技术特点
Gentry[2]	否	压缩解密电路
BGV[4]	是	模数变换、比特提取
CKKS[10]	是	模提升、模约减的近似同态操作
GSW[7]	否	同态累加、最高位比特的同态提取
FHEW[8]	是	自举时可将任意函数作为查找表进行计算, 可编程自举
TFHE[9]	是	可编程自举

Brakerski[5] 在基于 LWE 问题构建的方案中提出基于 q/B 不变(比例不变性 (scale invariance))的想法降低了噪声增长速度, 替换模数变换, 并延续了文献 [4, 20] 中的密钥变换技术. Fan 等人[6] 改进了文献 [5] 中的密钥变换技术, 引入了 2 个优化版本的密钥变换, 提高了计算速度 , 并提出了文献 [6] 的 RLWE 版本, 文献 [6] 也被称为 FV 方案. 现在的 BFV 方案是包含上述噪声优化技术的方案(比例不变性和密钥变换).

针对部分密文系数较大导致计算复杂度过高的问题, Bajard 等人[25] 提出了 RNS FV, 利用中国剩余定理(China remainder theorem, CRT)蕴含的一个余数系统(residue number system, RNS), 将具有较大系数的密文多项式分解为具有较小系数的 L 个残差多项式, 大幅降低密文系数过大产生的计算成本. Halevi 等人[26] 进一步优化改进, 提出 RNS 变体 HPS, 其具有更低的计算复杂性和噪声增长速度.

2017 年, Cheon 等人[10] 提出的 CKKS 的 LHE 方案, 由于基于 CKKS 方案的 LHE 版本不支持模运算且其解密电路需要模约减运算, CKKS 的解密电路为了有效解密初明文, 需保留噪声(计算误差). 所以, 对于 CKKS, 为了实现全同态的自举技术充满挑战. 1 年后, Cheon 等人[21] 将 CKKS 方案扩展为全同态加密方案, 并提出了扩大模数的技术, 并将其自举中的模约减函数简化为整数(或复数)上的多项式问题, 用三角函数近似模约减函数来降低计算成本. 但由于自举过程中的密文模数过大, 需要专门的大数计算支持, 为了降低这部分的计算开销, Cheon 等人[27] 提出了 CKKS 的 Full-RNS 变体, 通过快速基变换替换大整数运算, 将其同态运算的速度增加到原 CKKS

方案的4~10倍. Han等人[28]则在此基础上推广了CKKS的Full-RNS的变体, 并改进了自举中的正弦函数.

2.2 在布尔电路上实现的全同态加密方案

支持布尔电路的全同态加密方案主要是GSW[7], FHEW[8], TFHE[9], 它们均支持快速自举, 如表3所示.

GSW方案是Gentry等人[7]基于LWE问题和RLWE问题, 引入了近似特征向量法, 避免了密文维数增长. GSW方案没有评估密钥evk, 评估方可以在不知道用户公钥的情况下进行全同态计算. 因此, GSW能应用在多方计算(multi-party computation). 研究者们基于GSW方法提出了FHEW和TFHE方案, 目前TFHE方案的应用较广.

为进一步降低GSW方案的计算成本, 研究人员提出了不同优化的自举技术, 比如Alperin-Sheriff等人[32]提出了一种新的自举技术, 将解密视为算术函数而不是布尔电路; Hiromasa等人[29]构建了基于GSW方案的全同态方案, 支持包括同态矩阵运算在内的多种复杂同态运算.

Ducas等人[8]在Alperin-Sheriff等人[32]的工作基础上提出了FHEW方案, 并引入了一种同态计算2个LWE加密的NAND, 降低了噪声水平. 并且FHEW还采用了快速傅里叶变换(fast Fourier transform, FFT)加速同态乘法计算. 之后TFHE[9]使用更快的自举程序优化了这一方法.

相比于FHEW, TFHE优化了自举过程和自举密钥大小. TFHE方案提供了一种有效的方法来处理噪声密文, 但该方法需要知道明文的最高有效位, 这将导致较大开销. 此后Chillotti等人[30]改进了TFHE方案, 提出了2种打包方法和1种新的自举方法, 只需137 ms就可以将TLWE转换为低噪声的TRGSW的密文. 同时提出了无填充的可编程自举技术Wop-PBS, 降低了开销. Guimarães等人[31]则提出了可编程自举的多输出版本, 以达到自举执行1次时, 可以对不同变量执行多个同态操作.

为同态评估找到最佳密码参数集一直是应用FHE的障碍之一. Bergerat等人[33]基于TFHE方案解决了这一问题. 在该工作中, 每个FHE算子都附有噪声公式和成本模型, 以便于对噪声增长和执行时间的影响进行量化. 该工作将参数选择转化成优化问题, 并引入一个基于该优化问题的框架, 允许使用者在给定平面图(plain graph)、成本模型和噪声模型的情况下, 为类TFHE方案选择最佳FHE算法的相关参数.

2.3 小 结

本节介绍了近年来基于LWE问题的全同态加密方案在效率优化方面的工作. 总体来说, 全同态加密方案的主要效率瓶颈包括缓解噪声增长的自举操作以及缓解同态评估过程中密文维数快速膨胀所需的额外操作.

当前的全同态方案中需要使用自举操作来降低噪声, 从而实现无限次的同态计算, 但自举操作计算成本较高. 增加级数L的值可以在一定程度减少FHE中自举操作的次数, 但这也要求相应增大模数q. 然而, 过大的模数q会增加计算开销和存储需求. 因此, 如何在参数L和自举操作次数之间取得平衡, 以达到最优计算效率, 是当前学术界研究的重要问题之一.

另一方面, 重线性化等缓解密文维数膨胀问题的技术是算术电路全同态方案中的技术, 而算术电路支持SIMD批处理这一能有效提高计算效率的操作. 布尔电路上实现的全同态加密方案TFHE不需重线性化操作, 并且具备高效的自举能力, 但不支持SIMD批处理操作, 因此TFHE的效率有时不如其他方案.

目前基于LWE构建的FHE方案尚未同时实现快速自举和支持批处理操作. 因此, 基于其他困难问题构建的FHE方案也成为了广泛探索的研究方向, 例如, Dorõz等人[34]在IACR 2018会议上提出了基于有限域同构问题的FHE方案, 展现了FHE方案具有更多的可能性.

3 全同态加密算法库

全同态加密算法库实现了FHE方案的底层加密操作以及自举功能或相关的优化操作的API, 基于算法库提供的编程框架和接口, 用户能使用熟悉的高级程序语言构建运行在CPU或GPU上的FHE应用. 全同态加密算法库降低基于FHE的应用程序的开发成本, 推动FHE的应用.

全同态加密算法库除了提供关键加解密操作的API外, 大多数算法库提供例如: Lattigo[35]提供Go语言的BFV和CKKS方案, pyFHE[36], PySEAL[37], Pyfhel[38]框架提供Python接口的FHE操作. HElib[39], PALISADE[40], HEAAN[41], SEAL[42], Lattigo[35], FHEW[43], TFHE[44], FV-NFLlib[45]等算法库提供基于CPU上的执行优化. cuFHE[46]和nuFHE[47]计算库提供NTT或者FFT的CUDA实现, 方便在GPU平台实现FHE方案.

HElib[39]是2013年发布的全同态加密算法库, 由Shai等人开发, 使用NTL库[48]进行底层数学运算. 该库最初为BGV方案设计, 最新版本也支持CKKS方案, 提供LHE操作和BFV自举操作. 由NJIT和Duality

Technologies 共同开发的 PALISADE[40] 支持 BFV, BGV, CKKS, TFHE 多种方案, 其中基础数学运算基于数学加速库 NTL 实现. OpenFHE[23] 由 Duality Technologies 主导研发, 于 2022 年发布, 结合了 PALISADE, HElib, HEAAN 等项目的设计思想, 支持所有主要的 FHE 方案, 并提供剩余数系统算法的高性能实现. 同时, OpenFHE 提供对不同硬件平台例如 CPU 中高级矢量扩展(AVX)、图形处理单元(GPU)、现场可编程门阵列(FPGA)和专用集成电路(ASIC)的支持.

FV-NFLlib[45] 是一个由 C++编写的支持 BFV 方案的全同态加密算法库. 它在一个理想格密码学的数学库 NFLlib 基础上开发, 它所实现的 BFV 方案中采用了双 CRT 表示、NTT 等优化技术.

2015 年发布的简单加密算法库[42](the simple encrypted arithmetic library, SEAL)由微软研究院开发. SEAL 实现了 BFV 和 CKKS 方案, 大部分 API 对这 2 种方案通用, 提供 LHE 操作, 但不为任意方案提供自举操作. 随后, 第三方开发了 SEAL 的 Python 接口 TenSEAL[49], TenSEAL 能轻松集成到流行的机器学习框架(PyTorch 或者 Tensorflow)中, 支持直接在加密张量上执行操作, 以及点积、向量乘法、矩阵乘法等机器学习常用计算. 据工作[49] 介绍, TenSEAL 能够在不到 1 s 的时间内完成一次加密卷积网络的计算.

TFHE(the fast fully homomorphic encryption library)是由 Ilaria 等人[44] 在 2016 年提出, 基于 TFHE 算法, 提供逐门自举(gate-by-gate bootstrapping)操作. Zama[50] 于 2020 年提出基于 Rust 开发的算法 CONCRETE, 目前开源的是 CPU 版本, 即将推出 GPU 与 FPGA 版本. TFHEpp[51] 是在 CPU 上对 TFHE 进行完整 C++实现的全同态算法库, 比原始的 TFHE 稍快, 支持自举. cuFHE[46] 是 TFHE 的 GPU 算法库, 实现了多个逻辑门并行运行, 计算效率相比于 TFHE pp 有非常大的提升. 但是, cuFHE 库没有提供自举操作, 它虽然支持快速数论变换(NTT), 但为了调用 GPU 乘法指令, NTT 中的操作数位宽需要手动设置. 结果, cuFHE 在实际应用时性能低于预期.

表 4 中列举了主要的全同态加密算法库. 全同态加密方案虽然都是基于(R)LWE 困难问题构建, 但它们支持计算的数据类型不同, 密文编码、自举操作以及噪声管理等操作的实现方法不同. 以 BGV 和 BFV 为例, 它们支持整数域上的计算, 但 BGV 方案将消息编码在$\mathbb{Z}_Q$中整数的最低有效位, BFV 方案将消息编码在$\mathbb{Z}_Q$中整数的最高有效位. 同时它们两者采用了不同的噪声管理方法. 因此, 一个支持所有的加密方案并能根据用户安全需求选择合适加密方案的全同态加密算法库成为研究热点.

Table 4 Programming Language and Supported Schemes for FHE Algorithm Library

表 4 全同态加密算法库程序语言及其支持方案

库	语言	方案
HElib[39]	C++	BGV, CKKS
PALISADE[40]	C++	BGV, BFV, FHEW, TFHE, CKKS
HEAAN[41]	C++	CKKS
OpenFHE[23]	C++	BGV, BFV, FHEW, TFHE, CKKS
FV-NFLlib[45]	C++	BFV
SEAL[42]	C++/C#	BFV, CKKS
TenSEAL[49]	Python	BFV, CKKS
pyFHE[36]	Python	BFV, CKKS
PySEAL[37]	Python	BFV, CKKS
Pyfhel[38]	Python	BFV, CKKS
Lattigo[35]	Go	BFV, CKKS
FHEW[43]	C++	FHEW
TFHE[44]	C++/C	TFHE
CONCRETE[50]	Rust	TFHE
TFHEpp[51]	C++	TFHE
cuFHE[46]	Cuda/C++	TFHE
nuFHE[47]	Python	TFHE

Gouert 等人[52] 在 2023 年隐私增强技术研讨会(PETS)中发表的工作中指出部分流行的全同态加密算法库的优缺点以及最适合每个计算域的应用程序类型. 该工作采用了一系列被称为 Terminator 2 Benchmark Suite 的基准测试, 同时为了保证公平比较, 开发了一个多功能编译器(T2). T2 可以使用 5 个全同态加密算法库(HElib, Lattigo, PALISADE, SEAL, TFHE)作为后端将任意算法编译为其相应的加密形式, 并使用 3 种不同数据类型运行 T2 基准测试, 分别为 8 B 二进制、整数和浮点数. 对比二进制编码和整数编码效率时, 除 HElib 采用 BGV 方案外, 其他算法库使用 BFV 方案. 实验中, 每个库的 FHE 方案的安全参数至少 128 b.

表 5 列出各个算法库在乘法(Mult)和旋转(ROT)操作上的延迟结果. 该工作的实验结果表明, 在二进制域, TFHE 在速度和密文大小都是最优, 但对整数和浮点数明文的支持不如 HElib. PALISADE 在多核执行的优化使得它在整数和浮点数算术的加密速度最快. 由于 TFHE 不支持打包操作, 当用户想要并行

Table 5 Ciphertext Size of Integer, Binary and Floating-Point Arithmetic and the Delay of Multiplication and Rotation Operations

表 5 整数、二进制和浮点算术的密文大小以及乘法和旋转操作的延迟

库（缩写）	二进制编码①				整数编码②				浮点编码②			
	延迟时间/s		时间④/s	密文大小③/MB	延迟时间/s		时间⑤/s	密文大小③/MB	延迟时间/s		时间⑤/s	密文大小/MB
	Mult	ROT			Mult	ROT			Mult	ROT		
HElib v2.2.1（H）	< 1 000	< 0.1	12.1	49.4	< 10	< 0.1	1.5	6.2	< 1	< 0.1	1.58	9.5
Lattigo v3.0.2（L）	< 1 000	< 0.1	58.7	50.3	< 1	< 0.1	0.2	6.3	< 1	< 0.1	0.25	9.4
PALISADE v1.11.6（P）	< 10 000	< 0.1	36.4/125	14.7	< 1	< 0.1	0.2	**1.8**	< 10	< 0.1	0.46	5.8
SEAL v4.0（S）	< 1 000	< 0.1	109	14.6	< 1	< 0.1	0.4	**1.8**	< 1	< 0.1	0.65	**4.5**
TFHE v1.0.1（T）	< 100	< 0.1	4.0	**0.02**	-⑥	-	3 083	-	-	-	-	-
性能最优统计	T	T	T		P	P L	P L		P	P L	L	

注：①二进制运算使用 32 000 次分圆多项式，TFHE 使用 1 024 次分圆多项式. ②整数域和浮点数域运算，使用 16 000 次分圆多项式. ③使用每个库的最小参数来加密 8 b 数字并在 128 b 安全性下实现深度 10. ④使用二进制编码测量字大小为 8 B 的 CRC-8 基准测试在二进制域中的 FHE 总执行时间，PALISADE 的 36.4 s 为多线程实现的执行时间，125 s 为单核执行时间. ⑤针对整数域和浮点数域测量 LR 基准测试的总执行时间，且基准测试主要包括低深度的加法和乘法，TFHE 使用字长 $w = 16$ b. ⑥“-”表示暂无数据.

评估基准的多个实例时，HElib 是最佳选择.

全同态加密算法在飞速发展，不断降低全同态加密算法存储和计算成本. 除了支持全同态加密算法的 API，全同态加密算法库应继续考虑硬件特性，从执行和调度方面提供更优化的实现. 基于算法复杂性和硬件特性，算法库自动为应用提供合适的参数选择也成为目前开发库时关注的主要问题之一.

4 全同态硬件加速研究动态

硬件加速对于 FHE 的实际应用至关重要. 在 BGV, BFV, CKKS, TFHE 等典型 FHE 方案存在大量的多项式运算. 由于加密数据通常由 1 对多项式组成，且其系数可达数百或数千位，需要昂贵的多字运算. 这对多项式乘法来说，增大了计算的复杂性. 尤其是在执行大规模加密计算时，其性能相对于未加密的明文计算存在显著的性能差距. 例如，使用明文数据（11 982 个样本，196 个特征）训练一个逻辑回归（logistic regression, LR）模型进行 30 次迭代，CPU 需要花费大约 1.05 s. 然而，相同的 LR 模型在使用加密数据进行训练时，在相同的 CPU 上需要约 124 min，速度减少了约 99.99%. 此外，训练中使用的加密数据（198 MB）比明文数据（1.3 MB）大 152 倍. 为了提升 FHE 计算的性能，推动隐私保护计算的发展和应用，设计高效的硬件加速器成为研究的重要方向.

目前，研究人员要么加速全同态在 GPU 平台上的执行，要么关注 FPGA、ASIC、存算一体（PIM）的硬件加速器设计. 一方面针对全同态方案中关键的特定算子进行了优化，并充分挖掘硬件资源从而提高计算并行性；另一方面，针对全同态加密中数据量非常大的问题，设计数据访存优化或者采用存算一体架构缓解全同态加密中的访存瓶颈.

多项式乘法是全同态加密方案中最耗时的操作之一，为了提升执行全同态计算的硬件效率，大量同态运算被转化成为（I）NTT 或（I）FFT，其中大量工作聚焦（I）FFT 或（I）NTT 算子的优化. BGV, BFV, CKKS 需要（I）NTT 算子，而 TFHE 无需模约减，需要（I）FFT 算子. 因此在 FHE 硬件加速相关工作中针对 NTT 算子的优化工作，以针对 BGV, BFV, CKKS 的硬件加速器最为突出. NTT 可以被看做将问题“转换”为在有限域上的 FFT 变体，NTT 和 FFT 的基本操作相同，所以一些针对 FFT 的优化技术在 NTT 优化中适用. 比如 NTT 同样采用 Cooley-Tukey 或者 GS 优化算法. 目前支持 TFHE 的硬件加速器工作较少，表 6 总结了 2020—2023 年部分硬件加速器中 NTT 单元中的优化设计.

表 7 总结了不同类型硬件上全同态加密加速的特点. 接下来本文将分别介绍不同类型全同态加密硬件加速器.

4.1 全同态加密 GPU 加速研究

FHE 中多项式乘法是核心计算，多项式的次数和系数都是大数（如次数为 2^{16}，系数位宽为 2 000）. FHE 中的运算是通过大数计算或者 NTT 实现. 此外，有不少工作利用 RNS 对系数进行分解后，利用 GPU 的众核并行架构，提高 FHE 并行度和执行效率.

2013 年 Wang 等人[63] 对 Gentry 等人[64] 提出的 FHE 机制进行了 GPU 实现. 针对同态中大规模模乘的问题，Wang 等人[63] 采用了基于 Strassen 的 FFT 乘法及巴雷特归约以加速模归约. 实验表明，相比于 CPU，

Table 6 Optimization Work on NTT for Partial Hardware Accelerators from 2020 to 2023

表 6 2020—2023 年部分硬件加速器关于 NTT 的优化工作

硬件类型	加速器	NTT 优化方向
GPU	TensorFHE[53]	充分利用 TCU（tensor core unit）
FPGA	HEAX[54]	实现吞吐量可调，全流水线架构
	HEAWS[55]	为实现蝶形计算优化架构, 避免内存访问冲突
	Poseidon[56]	减少模计算
	FAB[57]	针对 NTT 数据路径进行优化
	Medha[58]	使用迭代 NTT 的方法, 消除了部分系数排列的需要
ASIC	F1[24]	产生向量友好的分解, 实现全流水线的 NTT 单元
	CraterLake[59]	在 F1 工作的基础上, 优化向量分解和数据流
	BTS[60]	使用系数级并行, 优化了数据路径
PIM	MeNTT[61]	减少数据流
	CryptoPIM[62]	更快的内存乘法，高吞吐量

GPU 方案(英伟达 GTX690)在加密和解密过程分别实现 174 倍和 7.6 倍的加速. 然而, Wang 等人[63] 指出随着密码参数规模增大时, 内存访问可能成为整体性能的瓶颈, 这将削弱计算优化带来的加速效果.

Al Badawi 等人[65] 于 2018 年基于 cuHE 提出了一个基于 CUDA 的 FV 同态加密方案. 该方案实现了 FHE 操作完全运行在 GPU 平台上, 无需将计算卸载到 CPU, 避免了跨平台的通信开销. 该方案使用 NTL 初始化加密密钥使得所有密钥存储在 GPU 中, 并且在 GPU 上进行 CRT/RNS 域的计算. 该工作在 Nivdia Tesla K80 GPU 和 CUDA 8.0 版本库实现了 80 位安全级别和大于 5 层的乘法深度的 FV 密码方案. 实验结果显示, 相比于基于微软 SEAL 算法库(CPU 平台的 C++实现)的实现, 该方案执行 FV 的性能具有明显的提升.

韩国首尔国立大学 Jung 等人[66] 实现了第 1 个 CKKS 自举的 GPU 实现. Jung 等人[66] 对同态乘和密钥交换的关键操作的分析发现, 这些操作的计算密度偏低, 内存访问是效率瓶颈. Full-RNS CKKS 的分解数 dnum 过大, 会增大存储需求. Jung 等人[66] 工作中提出选取适中的 dnum 值, 并采用算子融合和主函数重排等优化技术, 最大限度地利用 GPU 的并行计算能力来加速 FHE 操作.

中国科学院信息工程研究所的学者[53] 在 HPCA-2023 的工作中提出了 TensorFHE, 其基于高端 GPU (A100)的全同态加密加速实现. 该研究关注以利用张量核心单元(tensor core units, TCU)增强 NTT 的计算为目标, 通过最大化计算资源的利用率来加速 NTT 核函数的执行. 用矩阵向量乘法实现 NTT 以充分利用张量核. 同时 TensorFHE 通过优化数据布局, 实现了批量 FHE 操作时的计算吞吐提升. 实验表示, 针对于同态乘法和同态加法这 2 个关键算子, TensorFHE 相比之前工作取得了 397.1 倍和 1 035.8 倍的性能提升.

Table 7 Supported Encryption Schemes and Their Characteristics for FHE Hardware Accelerators

表 7 全同态加密硬件加速器支持加密方案及其特点

类型	加速器	支持的加密方案	特点
GPU	文献 [63]	Gentry 等人[64]	优化的 FFT 加速百亿比特级模乘
	文献 [65]	BFV	基于 CRT，DGT，RNS 多项式乘法的 GPU 实现
	Over 100x[66]	CKKS	算子融合和主函数重排的内存优化技术
	TensorFHE[53]	CKKS, BGV, BFV	有效利用 TCU, 优化 NTT
	GME[67]	CKKS	微架构扩展与优化
FPGA	Roy 等人[68]	FV	BFV 算子优化和多项式乘的并行优化技术
	HEAX[54]	CKKS	模计算和 NTT 优化技术
	HEAWS[55]	FV	面向云 FGPA 的实现；软硬件接口通信优化技术
	XHEC[69]	TFHE	对多比特计算优化加速，提出布尔门之间的并行技术
	Poseidon[56]	CKKS	计算算子拆分，算子的分时复用和重组合
	FAB[57]	CKKS	关键路径的计算并行，计算调度已提高数据重用
	FxHENN[70]	CKKS	资源限制下的设计空间探索技术，在低功耗 FPGA 上验证
	Medha[58]	CKKS	RNS 多项式计算单元；通过最少数量网络实现并行单元之间的互连
ASIC	F1[24]	BGV，GSW，CKKS	基于静态调度策略并减少数据移动，仅适用于浅乘法深度的子集
	CraterLake[59]	BGV，GSW，CKKS	解决了深度计算的开销
	BTS[60]	BGV，BFV，CKKS	第 1 个以自举为目标的加速器，可以实现无限的乘法深度
	MATCHA[71]	TFHE	快速且节能的加速器，通过近似方法加速多项式乘法
	ARK[72]	CKKS	软硬协同设计，大大提升自举效率
	SHARP[73]	CKKS	首次设计了短字长加速器缓解 FHE 中内存瓶颈和数据通信问题
PIM	CiM-HE[74]	BGV，BFV，GSW，TFHE，CKKS	第 1 个支持 BFV SHE 方案所有基本评估操作
	MemFHE[75]	FHEW	第 1 个客户端和服务器加速器；第 1 个以内存为中心的架构

2023 年 Shivdikar 等人[67] 在 ISCA 发表了 GME, 它在 AMD CDNA GPU 架构上引入微体系结构扩展和编译优化, 并利用了云中已建立的 GPU 生态系统. 该工作利用片上存储之间的互连提高数据访问效率. 同时使用了特定的 MOD 计算单元支持模运算. 相比早前的 GPU 实现, HE-LR 和 ResNet-20 中 FHE 的工

作负载平均加速了 14.6 倍，总体来说，优化工作减少了 38% 的冗余计算，减轻了 DRAM 的内存压力. 该工作探索了 LDS 大小对 FHE 工作负载性能的影响，发现当 LDS 大小增加到超过 15.5 MB 后并不会带来显著加速，此时 DRAM 带宽成为了主要瓶颈.

表 8 对比了部分 GPU 加速器的性能. 这些研究表明，GPU 相对较高的并行计算能力成为它的主要优势，但较小的片上存储和相对较低的 DRAM 访存成为 GPU 平台的硬件加速全同态的瓶颈. 此外，FHE 具有核心算法之一的模计算、大量整数算术运算、内存访问模式不规则的特点都造成 GPU 硬件利用率降级. 近年来，GPU 优化 FHE 的方法均为提升硬件利用率，降低冗余计算，有效利用内存有限带宽或者采用更先进的 HBM 内存.

Table 8 Comparison of Partial GPU Accelerator Performance

表 8 部分 GPU 加速器性能对比

GPU 型号	GPU 加速器	字长/b	功耗/W	FHE 加速单元性能/μs	
				Mult	ROT
V100	Over 100x[66]	54	250	2 960	2 550
A100	TensorFHE[53]	32	400	1 131	1 008
MI100	GME[67]	54	300+107.6①	464	364

注：①基于 CDNA 架构的 MI100GPU 功耗并未透露，原文作者显示公开可用的近似值.

4.2 全同态加密 FPGA 加速研究

FPGA 由于具有灵活性、低功耗和可扩展性等特点，被广泛配置在云服务器中，以提供定制化的加速. 近年来，国内外研究学者深入研究了基于 FPGA 的全同态计算加速器.

比利时鲁汶大学的 Roy 等人[68] 在 HPCA2019 发布一篇面向同态计算 FPGA 加速器的文章，加速的算法是不包含自举步骤的 SHE. 该文在 Arm+FPGA 异构平台上设计了一种加速加密数据同态计算的领域专用架构，其中 Arm 处理器用于执行不同的同态应用. 针对多项式乘法，通过并行计算核以及电路级、块级流水策略提高计算吞吐，并通过优化片上缓存来缓解片外访存压力. Roy 等人[68] 在 Xilinx Zynq UltraScale+ MPSoC ZCU102 开发套件上验证了该 FPGA 加速器设计的有效性. 此后 Roy 等人[76] 提出了 HEPCloud，该工作使用参数安全性更高的同态加密机制进行加速设计，并采用了 NTT 算法和巴雷特算法等来加速多项式计算和模规约等操作. 在此之后，比利时鲁汶大学研究学者设计了针对特定领域的协处理器架构 HEAWS[55]，该架构基于 Amazon AWS 云的高性能 FPGA（Xilinx Virtex UltraScale+系列），通过并行执行同态计算和流水线处理，以及优化软硬件通信接口效率来实现计算加速. 在深度为 4 的同态计算任务中，HEAWS 执行同态乘法每秒 613 次，并在基于全同态的神经网络任务上实现了 5 倍的提速.

韩国首尔国立大学 Paek 等人[69] 于 2022 年提出面向 TFHE 的 FPGA 加速器——XHEC. TFHE 的计算基于同态布尔门（Tgate），该研究关注基于 Tgate 的 N 位运算的计算效率. 除了考虑到单个 Tgate 内操作的并行性，该工作考虑了 TFHE 中 N 位运算对应的多个 Tgate 之间的并行性. XHEC 利用 Tgate 内以及 Tgate 间的并行实现了整体性能的提升. 文献 [69] 的作者在 CPU-FPGA 混合架构上（英特尔通用 CPU 与 Xilinx Alveo U280 型号 FPGA）搭建的机器上进行了验证，并与 CPU 的 TFHE 计算库[44]、GPU 上基于 cuFHE 库的 TFHE[42] 以及 ASIC 工作[71] 进行了对比. 在 128 b 量子安全参数下，N 位运算的吞吐率以及每瓦吞吐率平均取得 2.43 倍及 12.19 倍的提升.

美国加州圣地亚哥分校和微软公司的研究学者[54] 合作的工作 HEAX，是第 1 个面向 CKKS 加密方案的 FPGA 加速器，提供了对模计算的加速和 NTT 的高通量实现. 考虑到 NTT 在不同部分变换数量的不一致性，设计了一个可以根据需要调整数据吞吐量的架构以节省资源. 该工作在资源数目不同的 2 种 FPGA 上进行了验证，与 CPU 的 SEAL 库相比，HEAX 取得了 2 个数量级的加速效果，但是这个工作尚未对自举操作进行讨论. HEAX 有较好的加速效果，但不可编程，且其流水线架构是专门为 RNS-HEAAN 键切换设计的. 中国科学院和中关村实验室学者[56] 合作提出 Poseidon，旨在改善硬件资源和带宽消耗. 该工作将全同态加密计算拆分更细粒度的算子，并通过算子的分时复用和重组合，在 FPGA 资源有限的情况下最大化全同态计算的执行效率. 在 FPGA Xilinx Alveo U280 验证了联邦学习常用的 3 种加密神经网络（逻辑回归、LSTM、ResNet-20）和 1 种自举算法，对比 CPU 和 GPU 分别取得 1 300 倍和 52 倍的性能加速. FAB[57] 实现了对 CKKS 方案自举操作的 FPGA 加速，并支持实用的 FHE 参数设置（$N = 2^{16}$, lb $q = 54$）. FAB 利用最新的自举算法，并根据硬件约束进行参数的选择和 FHE 操作优化，支持更实用的 FHE 参数. 该工作通过关键路径的并行、算子调度以提高数据重用从而实现计算加速. 对于云上在 8 个 FPGA 上执行的逻辑回归（LR）模型训练，FAB 的性能比 CPU 和 GPU 分别高 456 倍和 6.5 倍.

意大利格拉茨大学与韩国先进研究院的学者[58]在 Medha 中提出了利用分治法的 N 项环完成 $2N$ 项环计算的 Medha 算法, 从而灵活支持多种同态加密参数集. 在架构设计方面, 该工作构建了 RNS 多项式计算单元, 并通过最少数量网络实现并行单元之间的互连. 该工作在 Xilinx Alveo U250 FPGA 验证了基于 RNS 的 CKKS 加密机制的性能, 加密机制的参数集规模为 lb q = 438, N = 214 和 lb q = 546, N = 215, 与微软 SEAL 库的实现对比, 该工作加速比分别达到了 68 和 78.

山东大学的学者 Zhu 等人[70]提出了 FxHENN, 一种面向同态加密的 CNN 推断(HE-CNN)计算的加速框架. FxHENN 通过设计空间探索, 根据可用的 FPGA 资源为 HE-CNN 分配优化的资源和生成加速器电路, 本文采用的低功耗 FPGA 设备, 验证了在边缘端实现全同态加密 CNN 的可行性.

总体来说, FPGA 主要通过加速部分算子工作, 并使用通用主机处理器处理其他算子, 同时依赖主机处理器来排序操作. 这不仅导致了过多的数据移动, 也使得所构建的算子过于专业化(使用固定参数), 限制了算法多样性, 同时限制了加速效果.

4.3 全同态加密 ASIC 加速研究

相比于 GPU 和 FPGA, ASIC 能根据全同态加密的计算和访存特点设计定制化的硬件支持, 从而提供最优的效率. 近年来面向全同态的专用加速器的关注度逐渐提高.

Feldmann 等人[24]在 MICRO'21 上提出基于 ASIC 的全同态加速器 FHE——F1. 针对 FHE 中的特殊算子, F1 设计了专门的 NTT 单元和支持向量化执行的同态计算单元. 为了应对数据访问瓶颈, F1 提供了能直接管理多层次片上存储、scratchpad 和分布式寄存器文件、数据预取机制, 以及最大化数据重用. 此外, 为了降低开销, F1 没有控制指令, 计算是静态调度. 14nm/12nm 工艺下综合出来的 F1 设计有 151 mm^2, 并且有 64 MB 的片上存储和高达 1TBps 的访存带宽. 然而, F1 其深度不足以运行完整的自举过程. 由于现代深度神经网络的深度通常达到数十甚至数百层, 支持无限乘法深度仍是 ASIC 加速器亟需解决的问题.

Samardzic 等人[59]在 ISCA'22 提出 CraterLake, 一个支持无限乘法深度的全同态加速器. 为了满足巨大的计算需求, CraterLake 采用了带有特定计算单元的极宽(2 048 通道)向量单处理器架构, 设计了一种固定转置网络来缓解不同通道单元之间的数据传输开销. 在深度神经网络任务上, ResNet 在 CPU 上每次推理计算需要 23 min, 而在类似的面积和功率预算下 CraterLake 每次推理需要 250 ms.

Kim 等人[60]提出了 BTS, BTS 是专门针对自举优化的加速器, 同样支持无限的乘法深度, 主要针对自举加密数据的吞吐量进行了优化. 文献 [60] 的作者分析发现全同态操作中天然的并行性. BTS 将大量处理单元网格化组织, 片上网络用于处理单元之间的有特定模式的数据传输. 与 F1 相比, BTS 更好地利用了数据并行, 采用系数级并行来避免部分 PE 间的数据交换. 实验显示, BTS 在 ResNet-20 和逻辑回归上的执行速度比 CPU 提高了 5 556 倍和 1 306 倍.

Jiang 等人[71]提出 MATCHA, 使用积极的自举密钥展开(BKU)来加速 TFHE 门, 由于 TFHE 与其他方案相比具有容错能力, 因此在 MATCHA 中, 他们使用 FFT 和 IFFT 来加速乘法运算, 该运算也适用于自举. 与之前的加速器相比, MATCHA 将 TFHE 门处理吞吐量提高了 2.3 倍, 每瓦吞吐量提高了 6.3 倍.

2022 年MICRO 会议上,韩国首尔国立大学Kim 等人[72]提出了 ARK, 它是一个软硬件协同设计的工作. 一方面它优化了 FHE 算法, 减少密钥交换和运行时数据生成, 使运行时数据保留在片上. 另一方面, ARK 架构中的基本功能单元实现了基变换、NTT 以及自同构操作. 为了减轻数据搬运开销, ARK 采用了数据分布策略并将芯片分为 2 个逻辑阈. 通过算法和架构设计协同设计, 7 nm ARK 的面积有 418.3 mm^2, 在乘法吞吐量上的表现超过了 F1 加速器 2 353 倍.

2023 年韩国首尔国立大学 Kim 等人[73]在 ISCA 发表的 SHARP 中设计了短字长(36 b)加速器来缓解 FHE 中内存瓶颈和数据通信问题, 此外 SHARP 针对(I)NTT 设计了层次结构, 同时包括定制的内存结构和计算单元. 与 ARK 类似, SHARP 设计了针对减少容量的体系结构和软件优化. 相比 ARK, SHARP 的面积仅为 178.8 mm^2, 提供了 1.57~11.5 倍的性能提升, 芯片面积减少了 125%~234%, 平均功耗降低了 115%~168%.

由上可以看出, ASIC 硬件加速器可以实现全同态计算的显著加速, 专用加速器除了针对 FHE 方案中的技术瓶颈(自举、TFHE 门)设计专门的加速架构, 更多的优化技术都是为了减少数据移动, 提高数据重用效率. 表 9 展示了 ASIC 加速器使用的资源对比. 图 4 展示了 F1 与 CraterLake 的加速架构对比.

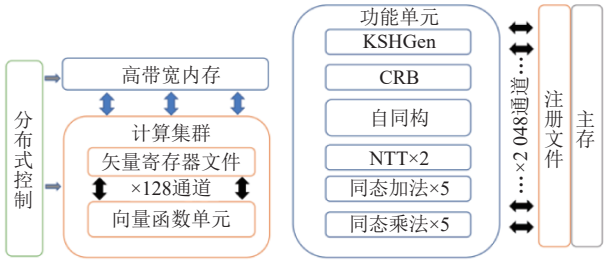
4.4 全同态加密存内计算加速研究

面向全同态加密算法存在典型内存受限(memory-bound)问题. 存内计算架构通过在存储器内完成计算,

Table 9 Resources Utilized in ASIC Accelerator

表 9 ASIC 加速器使用的资源

ASIC	年份	工艺节点/nm	片上存储/MB	面积/mm²	访存带宽/TBps	功耗/W
F1[24]	2021	12/14	64		1	91.0
CraterLake[59]	2022	12/14	256	472.3	1	317.0
BTS[60]	2022	7	512	373.6	1	163.2
ARK[72]	2022	7	512	418.3	2	281.3
SHARP[73]	2023	7	180	178.8	1	



(a) F1架构 (b) CraterLake架构

Fig. 4 Partial fully homomorphic encryption ASIC acceleration architecture

图 4 部分全同态加密 ASIC 加速架构

能大幅缓解数据搬运导致的内存瓶颈问题. 因此, 面向全同态加密的存内计算加速研究也受到广泛的重视.

美国圣母大学研究人员 Reis 等人[74]提出 CiM-HE, 一个基于存内计算的架构支持 BFV 加密机制的 SHE 方案, 但它也同样适用于包含类似操作的其他加密方案, 比如 BGV, GSW 等. CiM-HE 的设计主要针对 BFV 中使用的多项式运算, 其核心计算单元是 6T-SRAM 的存内计算阵列, 并通过定制的外围设备完成 BFV 加密机制中多项式计算的基本算子构建同态运算. 由于在 CiM 上实现的多项式乘法依赖于 Karatsuba 乘法, 所以并未采用更高级的优化, 如 NTT 或 RNS. 该工作利用存内计算架构高并行的计算能力完成多项式系数间计算的并行. 与 Roy 等人[55]的工作相比, 其执行速度提高 2.2 倍, 能量效率提高 88.1 倍. Gupta 等人[75]提出 MemFHE, 它是基于 FHEW 加密机制的全同态加密计算全流程加速. FHEW 密码系统对内存有巨大需求, MemFHE 为 FHEW 密码系统提供了第 1 个以内存为中心的架构, 并使用高度流水线的架构来加速自举的瓶颈过程. 通过密文级和操作级别并行, 以及操作之间的流水线设计, MemFHE 将吞吐量提高 3 个数量级.

NTT 是基于(R)LWE 问题密码系统中占主导地位的过程, 因而它的优化工作也可用在 FHE 硬件加速中. 比如美国莱斯大学的 Li 等人[61]提出 MeNTT, 用 6T-SRAM 阵列和外围辅助电路完成计算. 通过数据映射策略减少了 NTT 和逆 NTT 阶段之间的数据交换, 从而大大简化了路由开销. 类似地, 美国加州大学欧文分校的学者 Najatollahi 等人[62]提出基于非易失存储器的 NTT 多项式乘法加速器——CryptoPIM. CryptoPIM 考虑到 NTT 中不规则数据问题, 定制化了一个固定功能的数据交换器, 使得多个 NTT 阶段中数据并行传输.

存内计算架构作为加速全同态新的途径, 通过在存储器内完成计算, 解决有限带宽下内存访问带来的性能瓶颈, 这对计算密集型的 FHE 来说具有极大的优势. 这些工作为进一步优化和加速全同态加密计算提供了新的思路.

4.5 其 他

美国波士顿大学和美国麻省理工大学合作发表在 MICRO 2023 的工作 MAD[77], 提出了面向内存和数据访问相关的优化技术, 旨在解决 CKKS 内存访问的性能瓶颈. MAD 所提技术能够应用在不同的硬件(GPU/ASIC)上. MAD 加速器通过提出缓存优化以及减少乘法和旋转操作中 ModDown 子操作数量等内存感知设计技术, 来减少自举的内存带宽需求. 这些优化工作使数据重用最大化, 并减少数据访问模式切换次数, 从而降低自举的 DRAM 访问次数, 提高系统的性能和效率. MAD 也与之前 ASIC 工作做了对比. MAD 加速器将 FHE 的自举算术强度提高 3 倍, 并在逻辑回归训练、机器学习推断等应用中实现性能提升, 同时降低了片上内存需求. 该工作还提出了一个 SimFHE 模拟器, 用来评估 CKKS 中计算和内存带宽需求.

4.6 小 结

全同态加密具有较高的隐私保护潜力, 近年来全同态加密硬件加速器飞速发展, 以实现全同态加密的部署和应用.

作为典型的数据密集型算法, 全同态加密方案由于密文大小和相关参数都非常庞大, 内存访问开销高, 导致计算密度非常低. 为了缓解内存访问瓶颈问题, 全同态加密加速器都专门优化了内存管理策略.

表 10 列举了部分 FHE 加速器在全同态加密应用的性能. 这 3 种不同工作负载分别是自举、基于全同态加密的 LR 和 ResNet-20. 从表 10 可以观察到, 相比于 GPU 和 FPGA, ASIC 硬件平台整体加速效果更佳. 尤其是在自举操作, 除 F1 之外的 ASIC 加速器均表现出 100 ms 以内的自举时间. ASIC 定制设计的特

性使其能够提供最大化的加速能力. 相比之下, FPGA具有较高的灵活性, 且兼具流水线和数据并行性, 因此在某些情况下对FHE的加速具有优势. 但FPGA的可编程资源相对有限, 工作频率较低, 限制了其性能竞争力. 比如表10中2023年FPGA加速器FAB在自举操作和LR的表现不如同年GPU加速器GME. GME利用已建立的GPU生态系统和新兴计算单元(如Tensor Core)实现了良好的性能表现. 然而, GPU在片上存储有限且频繁的内存访问仍然制约了全同态加密的整体性能.

Table 10 Comparison of Partially Fully Homomorphic Accelerator for Hardware Performace

表10 部分全同态加速器硬件性能比较

类别	工作	年份	自举时间/ms	LR/ms	ResNet-20/ms
GPU	Over 100x[66]	2021	328.25	775	
	TensorFHE[53]	2023	157	178	3 793
	GME[67]	2023	**33.63**	**54.5**	**982**
FPGA	Poseidon[56]	2023		**72.98**	2 661
	FAB[57]	2023	**92.4**	103	
ASIC	F1[24]	2021	*	1 024	2 693
	CraterLake[59]	2022	4.5	119.62	321
	BTS[60]	2022	58.9	39.9	1910
	ARK[72]	2022	**3.7**	7.717	125
	SHARP[73]	2023		**2.53**	**99**

注: "*"表示仅支持单槽自举, 不支持打包自举. 黑体数值表示此硬件类别中执行效率最高.

综上, 现有的全同态加速器都试图通过利用更先进的架构技术来构建效能更优的硬件加速器, 从而大大提高了加速效果. 然而, 内存访问瓶颈仍是一个较大挑战. 存内计算加速器利用其独特架构优势, 可以有效缓解这一问题. 因此, 未来存算一体加速器的工作重点将集中在开发专用于全同态加密算子的存内计算单元, 并且更好地利用存内计算的并行优势.

5 结 语

全同态加密方案在十几年的发展过程中取得了非常优秀的成果. 基于(R)LWE问题构建的密码系统在深度学习、图像分类和语音识别等领域的相关工作中已初显成效. 同时一些基于全同态加密的具体应用将要或已经被逐步推广. 比如, Martins等人[78]使用BGV方案在云中实现了强大的隐私和实用性能, 已经被实际部署在微软浏览器软件用作密码监视.

虽然目前FHE在现实生活中的应用范围不断扩展, 但也面临一些安全挑战. 其中之一是恶意服务器可以利用FHE的可延展性执行密钥恢复攻击, 导致FHE的机密性被破坏. 此外, 随着人工智能、大数据等技术的迅猛发展, 数据采集、数据分析等应用普及, 导致了云端数据量不断增大, 这些挑战对FHE性能提出了极高的要求.

目前各种FHE方案在噪声控制、密文规模管理等方面的技术有快速的进展. 然而全同态算法仍然有较大的优化空间, 如何为全同态密码方案选择适当甚至最优的密码参数是亟需解决的问题. 此外, 近几年还出现了基于其他困难问题的全同态方案.

近年来, 全同态加密硬件加速器方面也取得了突出进展, 尤其是针对同态乘法和模计算这2类核心算子的优化和改进. 同时, 不同平台的硬件加速器工作大多致力于减少数据迁移, 并提高并行计算能力, 以应对全同态加密算法性能上的内存限制. 以往工作表明基于存内计算架构的全同态硬件平台极具潜力, 尤其是基于SRAM的存内计算技术, 其成熟性和高存储密度使其有望成为构建全同态加密基础设施的关键技术.

作者贡献声明: 河人华负责完成文字、图表和参考文献的撰写和修订; 李冰主要负责硬件部分内容的撰写, 并完成了文字修订; 杜一博对硬件章节的相关内容进行了修订; 王颖提出了关于主题和题目的修改建议; 李晓维对摘要和题目提出了修改建议; 韩银和对本文的研究对象提出了修改建议.

参 考 文 献

[1] Cyberlands. IO. Top 14 cybersecuity breaches in China[EB/OL]. [2023-12-01]. https://www.cyberlands.io/topsecuritybreacheschina

[2] Gentry C. Fully homomorphic encryption using ideal lattices[C]//Proc of the 41st Annual ACM Symp on Theory of Computing. New York: ACM, 2009: 169−178

[3] Rivest R L, Adleman L, Dertouzos M L. On data banks and privacy homomorphisms[J]. Foundations of Secure Computation, 1978, 4(11): 169−180

[4] Brakerski Z, Gentry C, Vaikuntanathan V. (Leveled) fully homomorphic encryption without bootstrapping[J]. ACM Transactions on Computation Theory, 2014, 6(3): No. 13

[5] Brakerski Z. Fully homomorphic encryption without modulus switching from classical GapSVP[C]//Proc of the 32nd Annual Cryptology Conf. Berlin: Springer, 2012: 868−886

[6] Fan Junfeng, Vercauteren F. Somewhat practical fully homomorphic

encryption[J/OL]. Cryptology ePrint Archive. [2012-03-22]. https://eprint.iacr.org/2012/144

[7] Gentry C, Sahai A, Waters B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based[G]//LNCS: 8042: Proc of the 33rd Annual Cryptology Conf on Advances in Cryptology(CRYPTO 2013). Berlin: Springer, 2013: 75−92

[8] Ducas L, Micciancio D. FHEW: Bootstrapping homomorphic encryption in less than a second[G]//LNCS 9056: Proc of the 34th Annual Int Conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2015: 617−640

[9] Chillotti I, Gama N, Georgieva M, et al. TFHE: Fast fully homomorphic encryption over the torus[J]. Journal of Cryptology, 2020, 33(1): 34−91

[10] Cheon J H, Kim A, Kim M, et al. Homomorphic encryption for arithmetic of approximate numbers[G]//LNCS 10624: Proc of the 23rd Int Conf on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2017). Berlin: Springer, 2017: 409−437

[11] Dathathri R, Saarikivi O, Chen Hao, et al. CHET: An optimizing compiler for fully-homomorphic neural-network inferencing[C]//Proc of the 40th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2019: 142−156

[12] Shaikh M U, Adnan W A W, Ahmad S A. Sensitivity and positive prediction of secured electrocardiograph (ECG) transmission using fully homomorphic encryption technique (FHE)[C]//Proc of the 6th IEEE-EMBS Conf on Biomedical Engineering and Sciences (IECBES). Piscataway, NJ: IEEE, 2021: 292−297

[13] Kocabas O, Soyata T. Towards privacy-preserving medical cloud computing using homomorphic encryption[M]//Virtual and Mobile Healthcare: Breakthroughs in Research and Practice. Hershey, PA: IGI Global, 2020: 93−125

[14] Sun Xiaoqiang, Zhang Peng, Liu J K, et al. Private machine learning classification based on fully homomorphic encryption[J]. IEEE Transactions on Emerging Topics in Computing, 2018, 8(2): 352−364

[15] Liu Mingjie, Wang An. Fully homomorphic encryption and its applications[J]. Journal of Computer Research and Development, 2014, 51(12): 2593−2603 (in Chinese)
(刘明洁, 王安. 全同态加密研究动态及其应用概述[J]. 计算机研究与发展, 2014, 51(12): 2593−2603)

[16] Bai Lifang, Zhu Yuefei, Li Yongjun, et al. Research progress of fully homomorphic encryption[J]. Journal of Computer Research and Development, 2024, 61(12): 3069−3087 (in Chinese)
(白利芳, 祝跃飞, 李勇军, 等. 全同态加密研究进展[J]. 计算机研究与发展, 2024, 61(12): 3069−3087)

[17] Marcolla C, Sucasas V, Manzano M, et al. Survey on fully homomorphic encryption, theory, and applications[J]. Proceedings of the IEEE, 2022, 110(10): 1572−1609

[18] Regev O. On lattices, learning with errors, random linear codes, and cryptography[J]. Journal of the ACM, 2009, 56(6): No. 34

[19] Stehlé D, Steinfeld R, Tanaka K, et al. Efficient public key encryption based on ideal lattices[G]//LNCS 5912: Proc of the 15th Int Conf on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2009: 617−635

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

[21] Cheon J H, Kim A, Kim M, et al. Bootstrapping for approximate homomorphic encryption[G]//LNCS 10820: Advances in Cryptology (EUROCRYPT 2018). Berlin: Springer, 2018: 360–384

[22] Boura C, Gama N, Georgieva M, et al. Chimera: Combining ring-LWE-based fully homomorphic encryption schemes[J]. Journal of Mathematical Cryptology, 2020, 14(1): 316−338

[23] Al Badawi A, Bates J, Bergamaschi F, et al. OpenFHE: Open-source fully homomorphic encryption library[C]//Proc of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. New York: ACM, 2022: 53−63

[24] Samardzic N, Feldmann A, Krastev A, et al. F1: A fast and programmable accelerator for fully homomorphic encryption[C]// Proc of the 54th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO-54). New York: ACM, 2021: 238−252

[25] Bajard J C, Eynard J, Hasan M A, et al. A full RNS variant of FV like somewhat homomorphic encryption schemes[G]///LNCS 10532: Proc of the 23rd Int Conf on Selected Areas in Cryptography. Berlin: Springer, 2016: 423−442

[26] Halevi S, Polyakov Y, Shoup V. An improved RNS variant of the BFV homomorphic encryption scheme[G]//LNCS 11405: Proc of the Cryptographers' Track at the RSA Conf on Topics in Cryptology(CT-RSA). Berlin: Springer, 2019: 83−105

[27] Cheon J H, Han K, Kim A, et al. A full RNS variant of approximate homomorphic encryption[G]//LNCS 11349: Proc of the 25th Int Conf on Selected Areas in Cryptography. Berlin: Springer, 2019: 347−368

[28] Han K, Ki D. Better bootstrapping for approximate homomorphic encryption[G]//LNCS 12006: Proc of the Cryptographers' Track at the RSA Conf on Topics in Cryptology(CT-RSA). Berlin: Springer, 2020: 364−390

[29] Hiromasa R, Abe M, Okamoto T. Packing messages and optimizing bootstrapping in GSW-FHE[J]. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2016, 99(1): 73−82

[30] Chillotti I, Gama N, Georgieva M, et al. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE[G]//LNCS 10624: Proc of the 23rd Int Conf on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2017: 377−408

[31] Guimarães A, Borin E, Aranha D F. Revisiting the functional bootstrap in TFHE[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021: 229−253

[32] Alperin-Sheriff J, Peikert C. Faster bootstrapping with polynomial error[G]//LNCS 8616: Proc of the 34th Annual Cryptology Conf on Advances in Cryptology(CRYPTO 2014). Berlin: Springer, 2014: 297−314

[33] Bergerat L, Boudi A, Bourgerie Q, et al. Parameter optimization and larger precision for (T) FHE[J]. Journal of Cryptology, 2023, 36(3):

28

[34] Doröz Y, Hoffstein J, Pipher J, et al. Fully homomorphic encryption from the finite field isomorphism problem[G]//LNCS 10769: Proc of the 21st IACR Int Workshop on Public Key Cryptography. Berlin: Springer, 2018: 125−155

[35] EPFL-LDS, Tune Insight SA. Lattigo v5.0.2[EB/OL]. (2023-11-28)[2024-03-20]. https://github.com/tuneinsight/lattigo

[36] Erabelli S. pyFHE-A Python library for fully homomorphic encryption[D]. Cambridge, MA: MIT Press, 2020

[37] Titus A J, Kishore S, Stavish T, et al. PySEAL: A Python wrapper implementation of the SEAL homomorphic encryption library[J]. arXiv preprint, arXiv: 1803.01891, 2018

[38] Ibarrondo A, Viand A. Pyfhel: Python for homomorphic encryption libraries[C]//Proc of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography. New York: ACM, 2021: 11−16

[39] Shai H, Victor S. HElib 2.3.0[EB/OL]. (2023-07-18)[2024-03-20]. https://github.com/homenc/HElib

[40] Dave C, Kurt R, Yuriy P. PALISADE[EB/OL]. (2022-11-02)[2024-03-20]. https://gitlab.com/palisade/palisade-release

[41] Kim A. HEAAN v2.1[EB/OL]. (2017-09-05)[2024-03-20]. https://github.com/snucrypto/HEAAN

[42] Microsoft Research. Microsoft SEAL v4.1.1[EB/OL]. (2023-01-11)[2024-03-20]. https://github.com/Microsoft/SEAL

[43] Leo D, Daniele Mi. FHEW v2.0-beta[EB/OL]. (2017-05-31)[2023-04-25]. https://github.com/lducas/FHEW

[44] Ilaria C, Nicolas G, Mariya G, et al. TFHE v1.0.1[EB/OL]. (2017-08-16)[2023-05-27]. https://github.com/tfhe/tfhe

[45] CryptoExperts. FV-NFLlib[EB/OL]. (2016-07-26)[2023-04-06]. https://github.com/CryptoExperts/FV-NFLlib

[46] Wei Dai. cuFHE v1.0_beta[EB/OL]. (2018-03-14)[2023-09-11]. https://github.com/vernamlab/cuFHE

[47] NuCypher. nuFHE v0.0.3[EB/OL]. (2019-07-20)[2023-07-14]. https://github.com/nucypher/nufhe

[48] Victor S. NTL v11.5.1[EB/OL]. (2021-06-24)[2023-04-07]. https://github.com/libntl/ntl

[49] Benaissa A, Retiat B, Cebere B, et al. TenSEAL: A library for encrypted tensor operations using homomorphic encryption[J]. arXiv preprint, arXiv: 2104.03152, 2021

[50] Zama. CONCRETE: TFHE compiler that converts Python programs intoFHEequivalentv2.5.0[EB/OL].(2024-01-23)[2024-03-20].https://github.com/zama-ai/concrete

[51] Kotaro M. TFHEpp v8[EB/OL]. (2023-10-01)[2023-08-07]. https://github.com/virtualsecureplatform/TFHEpp

[52] Gouert C, Mouris D, Tsoutsos N. Sok: New insights into fully homomorphic encryption libraries via standardized benchmarks[J]. Proceedings on Privacy Enhancing Technologies, 2023, 2023(3): 154−172

[53] Fan Shengyu, Wang Zhiwei, Xu Weizhi, et al. TensorFHE: Achieving practical computation on encrypted data using GPGPU[C]//Proc of the 29th IEEE Int Symp on High-Performance Computer Architecture (HPCA-29). Piscataway, NJ: IEEE, 2023: 922−934

[54] Riazi M S, Laine K, Pelton B, et al. HEAX: An architecture for computing on encrypted data[C]//Proc of the 25th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020: 1295−1309

[55] Turan F, Roy S S, Verbauwhede I. HEAWS: An accelerator for homomorphic encryption on the Amazon AWS FPGA[J]. IEEE Transactions on Computers, 2020, 69(8): 1185−1196

[56] Yang Yinghao, Zhang Huaizhi, Fan Shengyu, et al. Poseidon: Practical homomorphic encryption accelerator[C]//Proc of the 29th IEEE Int Symp on High-Performance Computer Architecture (HPCA-29). Piscataway, NJ: IEEE, 2023: 870−881

[57] Agrawal R, de Castro L, Yang Guowei, et al. FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption[C]// Proc of the 29th IEEE Int Symp on High-Performance Computer Architecture (HPCA-29). Piscataway, NJ: IEEE, 2023: 882−895

[58] Mert A C, Kwon S, Shin Y, et al. Medha: Microcoded hardware accelerator for computing on encrypted data[J/OL]. Cryptology ePrint Archive. [2022-10-12]. https://eprint.iacr.org/2022/480

[59] Samardzic N, Feldmann A, Krastev A, et al. CraterLake: A hardware accelerator for efficient unbounded computation on encrypted data[C]//Proc of the 49th Annual Int Symp on Computer Architecture. New York: ACM, 2022: 173−187

[60] Kim S, Kim J, Kim M J, et al. BTS: An accelerator for bootstrappable fully homomorphic encryption[C]//Proc of the 49th Annual Int Symp on Computer Architecture. New York: ACM, 2022: 711−725

[61] Li Dai, Pakala A, Yang Kaiyuan. MeNTT: A compact and efficient processing-in-memory number theoretic transform (NTT) accelerator[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2022, 30(5): 579−588

[62] Nejatollahi H, Gupta S, Imani M, et al. CryptoPIM: In-memory acceleration for lattice-based cryptographic hardware[C/OL]//Proc of the 57th ACM/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE, 2020[2024-08-21]. https://ieeexplore.ieee.org/abstract/document/9218730

[63] Wang Wang, Hu Yin, Chen Lianmu, et al. Exploring the feasibility of fully homomorphic encryption[J]. IEEE Transactions on Computers, 2013, 64(3): 698−706

[64] Gentry C, Halevi S. Implementing Gentry's fully-homomorphic encryption scheme[G]//LNVC 6632: Proc of the 30th Annual Int Conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2011: 129−148

[65] Al Badawi A, Veeravalli B, Mun C F, et al. High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018, 2018(2): 70−95

[66] Jung W, Kim S, Ahn J H, et al. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021, 2021(4): 114−148

[67] Shivdikar K, Bao Yuhui, Agrawal R, et al. GME: GPU-based microarchitectural extensions to accelerate homomorphic encryption[C]//Proc of the 56th Annual IEEE/ACM Int Symp on

Microarchitecture. New York: ACM, 2023: 670−684

[68] Roy S S, Turan F, Jarvinen K, et al. FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data[C]//Proc of the 25th IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2019: 387−398

[69] Nam K, Oh H, Moon H, et al. Accelerating n-bit operations over TFHE on commodity CPU-FPGA[C/OL]//Proc of the 41st IEEE/ACM Int Conf on Computer-Aided Design. New York: ACM, 2022[2024-08-31]. https://dl.acm.org/doi/10.1145/3508352.3549413

[70] Zhu Yilan, Wang Xinyao, Ju Lei, et al. FxHENN: FPGA-based acceleration framework for homomorphic encrypted CNN inference[C]//Proc of the 29th IEEE Int Symp on High-Performance Computer Architecture (HPCA-29). Piscataway, NJ: IEEE, 2023: 896−907

[71] Jiang L, Lou Q, Joshi N. MATCHA: A fast and energy-efficient accelerator for fully homomorphic encryption over the Torus[C]// Proc of the 59th ACM/IEEE Design Automation Conf. New York: ACM, 2022: 235−240

[72] Kim J, Lee G, Kim S, et al. ARK: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse[C]//Proc of the 55th IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2022: 1237−1254

[73] Kim J, Kim S, Choi J, et al. SHARP: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption[C/OL]//Proc of the 50th Annual Int Symp on Computer Architecture. New York: ACM, 2023[2024-08-31]. https://dl.acm.org/doi/10.1145/3579371.3589053

[74] Reis D, Niemier M T, Hu X S. A computing-in-memory engine for searching on homomorphically encrypted data[J]. IEEE Journal on Exploratory Solid-State Computational Devices and Circuits, 2019, 5(2): 123−131

[75] Gupta S, Cammarota R, Šimunić T. MemFHE: End-to-end computing with fully homomorphic encryption in memory[J]. ACM Transactions on Embedded Computing Systems, 2024, 23(2): 1−23

[76] Roy S S, Järvinen K, Vliegen J, et al. HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation[J]. IEEE Transactions on Computers, 2018, 67(11): 1637−1650

[77] Agrawal R, De Castro L, Juvekar C, et al. MAD: Memory-aware design techniques for accelerating fully homomorphic encryption[C]//Proc of the 56th Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2023: 685−697

[78] Martins P, Sousa L. A methodical FHE-based cloud computing model[J]. Future Generation Computer Systems, 2019, 95: 639−648



He Renhua, born in 1996. Master candidate. Her main research interests include homomorphic encryption algorithm and homomorphic encryption accelerator.

河人华, 1996 年生. 硕士研究生. 主要研究方向为同态加密算法、同态加密加速器.



Li Bing, born in 1990. PhD, associate professor. Her main research interests include storage-compute fusion architecture's software and hardware design, lightweighting of deep learning models, and the reliablility and security of accelerators.

李　冰, 1990 年生. 博士, 副研究员, 主要研究方向为存算融合架构的软硬件设计、深度学习模型轻量化、加速器的可靠性及安全.



Du Yibo, born in 1999. PhD candidate. His main research interests include domain-specific processor and design automation.

杜一博, 1999 年生. 博士研究生. 主要研究方向为领域专用处理器、设计自动化.



Wang Ying, born in 1985. PhD, professor. His main research interest includes architectures of novel EDA, processor and memory system.

王　颖, 1985 年生. 博士, 研究员. 主要研究方向为新型 EDA、处理器与存储系统体系结构.



Li Xiaowei, born in 1964. PhD, professor. His main research interests include integrated circuit testing, EDA, fault-tolerant computing, and hardware security.

李晓维, 1964 年生. 博士, 教授. 主要研究方向为集成电路测试、EDA、容错计算、硬件安全.



Han Yinhe, born in 1980. PhD, professor. His main research interests include computer architecture and processor chips.

韩银和, 1980 年. 博士, 研究员. 主要研究方向为计算机体系结构、处理器芯片.