AKCN-MLWE 算法 AVX2 高效实现

杨昊[1)]刘哲[1)]黄军浩[1)]沈诗羽[2)]赵运磊[2)]

[1)](南京航空航天大学计算机科学与技术学院,南京 211106)

[2)](复旦大学计算机科学技术学院,上海 200433)

摘　要　随着量子计算机的快速发展,经典密码系统面临巨大的威胁.Shor 算法可以在量子计算机上多项式时间内分解大整数和求解离散对数,而这两类问题分别对应经典公钥密码系统中的 RSA 和椭圆曲线密码(ECC)所依赖的困难问题,因此可以抵御量子计算攻击的后量子密码近年来受到广泛的研究.格密码是后量子密码中最为高效且拓展性强的一类密码算法,在未来会逐步替代传统公钥密码算法(RSA、ECC 等).256 位高级向量扩展(AVX2)指令集是英特尔 64 位处理器中普遍支持的一类单指令多数据(SIMD)指令集,可用于并行计算.但是,由于格密码结构复杂,在支持 AVX2 指令集的英特尔 64 位处理器上难以对格密码方案进行高适配的深度优化.AKCN-MLWE 算法是我国自主设计的基于模格上容错学习(MLWE)问题的格密码密钥封装(KEM)方案,是中国密码学会举办的公钥密码算法竞赛第二轮的获奖算法.本文基于 256 位高级向量扩展(AVX2)指令集设计了针对 AKCN-MLWE 算法的高效实现方案,包括以下几个关键优化点:针对多项式乘法,本文结合最优的数论变换(NTT)算法,将 NTT 的最后一层转换为线性多项式并使用 Karatsuba 算法进行加速计算,大幅提升计算效率的同时减少了预计算表的空间占用;针对取模运算,本文结合了 Barrett 约减算法和蒙哥马利约减算法的优势,同时采用延迟约减技术降低取模次数;本文针对所有多项式运算均实现了高度并行化,设计了针对多项式压缩与解压缩的并行算法,进一步提升了实现效率.本文设计的 AKCN-MLWE 算法 AVX2 高效实现方案在八核 Intel Core i9-9880H 处理器上仅需不到 0.04 毫秒即可完成一次完整的 KEM(包括密钥生成、密钥封装和密钥解封装),相比于参考实现提升 8.84 倍,其中密钥生成提升 7.07 倍,密钥封装提升 7.90 倍,密钥解封装算法提升 11.78 倍.本文提出的 AKCN-MLWE 算法 AVX2 实现方案在相近经典经典安全强度下性能优于美国国家标准技术研究所(NIST)后量子密码标准化进程第二轮中众多格密码方案(Kyber、NewHope 和 Saber 等).同时,本文设计的部分优化方案可用于提升 Kyber、NewHope 等格密码方案的性能.

关键词　后量子密码;格密码;高级向量扩展;数论变换;模约减;多项式运算

中图法分类号　TP391

High-SpeedAVX2Implementation of AKCN-MLWE

YANGHao[1)]LIUZhe[1)]HUANGJun-Hao[1)]SHENShi-Yu[2)]　ZHAOYun-Lei[2)]

[1)](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing211106)

[2)](School of Computer Science, Fudan University, Shanghai 200433)

Abstract　As quantum computers developing rapidly, classical cryptosystems face a huge threat. Shor's algorithm can factor large integers and solve discrete logarithms in polynomial time on quantum computers, and these two types of problems correspond to the difficult problems relied on by RSA and elliptic curve cryptography (ECC) in classical public-key cryptosystems, respectively, so post-quantum cryptography that can withstand quantum computing attacks have been widely studied in recent years. Post-quantum cryptography is a

本课题得到国家自然科学基金青年项目(No.61802180)、十三五国家密码发展基金重点项目(No.MMJJ20180105)、密码科学技术国家重点实验室开放课题基金(No.MMKFKT201809)、江苏省自然科学基金(No.BK20180421)资助.杨昊,博士研究生,主要研究领域为公钥密码学、密码工程.E-mail: yang_hao@nuaa.edu.cn.刘哲(**通信作者**),博士,教授,主要研究领域为公钥密码学、密码工程.E-mail: zhe.liu@nuaa.edu.cn.黄军浩,硕士研究生,主要研究领域为公钥密码学、密码工程.E-mail: jhhuang_nuaa@126.com.沈诗羽,硕士研究生,主要研究领域为后量子密码、密码工程和密码协议.E-mail:syshen19@fudan.edu.cn.赵运磊,博士,教授,主要研究领域为后量子密码、密码协议和计算理论.E-mail: ylzhao@fudan.edu.cn.

collection of public-key cryptosystems which run on classical computers and are considered to be resistant to quantum attacks. Lattice-based cryptography was proposed as the most efficient and scalable type of post-quantum cryptography to resist quantum computer attacks, which will replace classical public-key cryptosystems (RSA, ECC, etc.) in the near future. The 256-bit Advanced Vector Extensions (AVX2) instruction set is a class of Single Instruction Multiple Data (SIMD) instruction set commonly supported in Intel 64-bit processors for parallel computing. However, due to the complex structure of lattice-based cryptography, it is difficult to perform deep optimization for lattice-based cryptographic schemes with a high degree of adaptability on Intel 64-bit processors that support the AVX2 instruction set. The AKCN-MLWE scheme is a lattice-based key encapsulation mechanism (KEM) scheme proposed by a Chinese research team, which is based on the module learning with errors (MLWE) problem. The scheme is also one of the winning algorithms of the public-key cryptographic algorithm competition organized by the Chinese Association for Cryptologic Research. In this paper, we present a high-speed implementation of the AKCN-MLWE scheme using the 256-bit advanced vector extension (AVX2) instruction set provided by Intel 64-bit processors. Firstly, we pick the state-of-the-art number theory transformation (NTT) algorithm, which reduces one layer of both forward and inverse NTT and transforms one-dimensional vectors in the NTT domain into linear polynomials. Therefore, we tweak the Karatsuba algorithm to calculate the linear polynomial multiplications in the NTT domain, which significantly improves the computational efficiency and reduces the space occupation of the precomputed table. Furthermore, we combine the advantages of the Barrett reduction and the Montgomery reduction and further use the lazy reduction technique to reduce the number of times that modular operations are called. Finally, we achieve a high degree of parallelization for all polynomial operations and design a parallel algorithm for polynomial compression and decompression, which further improves the efficiency of our AVX2 implementation of AKCN-MLWE. Our high-speed AVX2 implementation of AKCN-MLWE can complete a full key encapsulation mechanism (KEM, including key generation, key encapsulation, and key decapsulation) in less than 0.04 milliseconds on an 8-core Intel Core i9-9880H processor, which is 8.84 times faster than the reference implementation, including 7.07 times faster key generation, 7.90 times faster key encapsulation and 11.78 times faster key decapsulation. Our high-speed AVX2 implementation of AKCN-MLWE outperforms many lattice-based cryptographic schemes (Kyber, NewHope, and Saber, etc.) in the second round of the National Institute of Standards and Technology (NIST) post-quantum cryptography standardization process at similar classical security strength. Meanwhile, some of the optimization techniques in this article can be used to improve the performance of Kyber, NewHope, and other lattice-based cryptographic schemes.

Key words post-quantum cryptography; lattice-based cryptography; advanced vector extensions; number theory transform; modular reduction; polynomial operation

1 引言

公钥密码算法体制在互联网时代被广泛地应用到各种网络安全协议中,如 TLS,SSL 等.RSA 和椭圆曲线密码作为目前使用最广泛的两种传统公钥密码算法,它们所提供的密钥封装机制,密钥协商协议以及数字签名协议可为通信双方提供保密性,完整性以及不可抵赖性,被广泛应用到云计算,电子商务,金融行业等电子应用中.然而随着量子计算技术的不断发展,传统公钥密码算法体制遭受到极大的威胁,Shor 算法[1]首先提出在量子计算机中,现存的基于离散对数和大整数分解难题的公钥密码算法均可在多项式时间复杂度内被破解.因此,虽然传统公钥密码算法体制在传统计算机中是难以攻破的,但随着量子计算能力的不断增强,传统公钥密码算法体制在量子计算时代显然是不安全的.因此,能够抵抗量子攻击的新型公钥密码算法——后量子密码算法的研究成为目前学界的研究热点.

2016 年 12 月,美国国家标准与技术局(NIST)面向全球学者征集后量子密码算法标准,征集了大量的后量子公钥密码算法设计方案.在 NIST 举办的后

量子密码算法竞赛中,进入二轮候选公钥密码算法名单的共有 26 个.大多数候选算法可以归类为格,编码以及多变量三大类数学难题中,其中基于格的候选算法占了 12 个,格密码算法由于其在最差情况下也具有相当高的安全强度,快速的算法实现以及算法设计简洁性使其成为目前主流的后量子密码算法.2019 年由中国密码学会举办的全国密码算法设计竞赛也积极鼓励设计后量子密码算法,在竞赛中总共有 14 个获奖算法,其中有 11 个方案基于格上数学难题.AKCN-MLWE[2]作为中国公钥密码算法竞赛的获奖算法之一,算法安全性以及算法实现效率得到了中国密码学会以及评审专家的肯定和认可.

格密码算法通常是建立在 LWE(Learning with Errors)困难问题[3, 4]上的.记$\mathbb{Z}_q^{m\times n}$为m行n列的矩阵,其中每个元素均为 0 到q之间的整数.标准格的 LWE 困难问题是给定$(\mathbf{A},\mathbf{b})$和$\mathbf{b}=\mathbf{As}+\mathbf{e}$,其中$A\in\mathbb{Z}_q^{n\times n}$,$s,e,b\in\mathbb{Z}_q^{n\times 1}$,求$\mathbf{s}$是困难的.标准格的 LWE 困难问题涉及非常耗时的矩阵乘法运算,而且其密钥和密文存储代价非常大.为了降低计算时间,密钥和密文大小,Lyubashevsky 等提出了 RLWE(Ring Learning with Errors)困难问题[5],通过构建一个额外的多项式环$R_q=\mathbb{Z}_q[X]/(X^n+1)$将标准格中的$n$维向量替换成秩比较小的多项式,带来了很好的效率和存储上的提升.但由于 RLWE 引入了额外的代数结构,可能存在安全隐患,如最近提出的利用环理想格代数结构的攻击[6-8].为了在安全和性能之间作一个平衡,基于 MLWE[9]困难假设的设计方案在多项式环结构上引进了一个很小的维度k(通常为 2,3,4),并降低 RLWE 中的维度n来保证运行效率与 RLWE 相当.

AKCN-MLWE[2]是基于 MLWE 困难假设的密钥封装算法.AKCN-MLWE 的参数设计使得其适合使用数论转换技术对多项式乘法运算进行加速,而 NTT 又具有非常好的并行性,因此,可以利用并行技术对 AKCN-MLWE 进行优化实现.AVX2[1]作为英特尔处理器中一类比较强大的单指令多数据(SIMD)指令集,它可以同时处理 8 组 32 位数据或者 16 组 16 位数据,由于 AKCN-MLWE 中素数的位数是小于 16 位的,因此我们可以充分地利用 AVX2 对 AKCN-MLWE 中的底层运算进行加速.虽然目前英尔特处理器中已经存在比 AVX2 更新型的指令集,如 AVX-512,而且相对于 AVX2,AVX-512 的并行性和性能有显著的提升,利用 AVX-512 对密码算法进行优化实现将是未来的重要研究内容.但是由于 AVX2 指令集目前的普及率比 AVX-512 高,现存的大量参考文献中,大多数文献是利用 AVX2 对密码算法进行优化实现并作为性能对比的指标.因此,为了将 AKCN-MLWE 的实现性能与其它后量子密码算法标准进行对比,本文选择 AVX2 对 AKCN-MLWE 进行优化.但是,由于 AVX2 和 AVX-512 除了寄存器大小和部分指令集不同之外,其设计思路是类似的,因此,利用 AVX2 对 AKCN-MLWE 进行的并行实现思路可以推广到 AVX-512 并获得更优的实现性能,这部分扩展工作我们将在后续工作中进行.AKCN-MLWE 作为我国密码算法竞赛中的第二轮候选算法,在当前国家强调网络空间安全重要性的背景下,对我国学者自主研究的格密码算法进行优化实现具有重要的意义,可以为我国未来的后量子密码算法标准制定和工程实现积累经验,为我国制定具有安全自主可控的密码算法标准奠定基础.

本文针对我国密码算法竞赛中的二轮候选算法 AKCN-MLWE,充分利用 AVX2 强大的并行性,对其底层多项式乘法,模约减,多项式运算等进行优化实现研究.本文主要贡献有以下几点:1.多项式乘法模块采用删减一层的 NTT 并结合 Karatsuba 算法进行点乘,提升实现效率的同时降低了空间占用.2. 充分结合了 Barrett 约减算法以及蒙哥马利约减算法的优势来提升约减效率,同时使用延迟约减的方法最小化约减次数.3. 针对多项式加减、序列化等操作进行了高度并行的优化,进一步提升实现效率.

2 相关工作

NIST 后量子密码标准征集中要求各算法设计团队对本团队提交的算法进行优化,其中支持 AVX2 的处理器是最重要的优化平台之一.多数算法都相应地提交了基于 AVX2 的快速实现.NIST 第二轮候选算法中与 AKCN-MLWE 算法相关性较高的算法包括基于 RLWE 问题的 NewHope,基于 MLWE 问题的 Kyber.

Alkim 等人[10]在 2016 年公开了 NewHope 算法,为基于 RLWE 问题构造的密钥交换协议,该算法的模数为 12289,其中使用浮点型 NTT 进行多项式乘法,并且采用了蒙哥马利约减和 Barrett 约减相结合

1 Intel Corporation, Intel® 64 and ia-32 architectures software developer's manual, 2006

的延迟约减方法.

Longa[11]等人在 2016 年提出了针对 RLWE 中 NTT 的 AVX2 优化方案,其中使用标准形式的 NTT 以及针对特定模数的约减算法.

Seiler[12]在 2018 年提出了对 RLWE 的 AVX2 优化方案,其中包含了整形 NTT 和多种模约减算法.

Bos 等人[13]在 2018 年发表了 Kyber 算法的论文,为基于 MLWE 问题的密钥分装方案,同时为 NIST 第一轮候选算法.该算法可以看作是基于 NewHope 算法的后继算法,其中使用 Seiler 文章中的整形 NTT 进行多项式乘法,其模数与 AKCN-MLWE 相同,为 7681.

Lyubashevsky 等人[14]在 2019 年提出了 NTTRU,对 NTRU 的参数进行了调整使之可使用 NTT 加速,其环为 $\mathbb{Z}_{7681}[X]/(X^{768}-x^{384}+1)$.其中用到的 NTT 技术即为 Seiler 文章中的方法.

在 2019 年 NIST 第二轮中,Kyber 算法的参数进行了调整,得益于采用删减一层的 NTT 算法,其模数由 7681 降低为 3329,同时结合了教科书式的线性多项式乘法,针对其新模数 3329 同样使用了延迟约减方法.本文使用了其删减一层的 NTT 技术.

3 背景知识

3.1 符号表示

环: 令 $R=\mathbb{Z}[X]/(X^n+1), R_q=\mathbb{Z}_q[X]/(X^n+1)$,其中$q$是素数,$n=2^{n'-1}, X^n+1$表示第$2^{n'}$个分圆多项式在 AKCN-MLWE 中,$n=256, n'=9, q=7681$.正常字母表示$R, R_q$中的元素,粗体小写字母表示每项系数为$R, R_q$中的元素的向量,粗体大写字母表示矩阵,所有向量均表示列向量,使用$\mathbf{v}^T$或$\mathbf{A}^T$表示向量或矩阵的转置.

分布:对于一个集合S,令$s \leftarrow S$表示s是一个均匀分布在集合S上的元素.

可扩展输出函数:令Sam表示一个可输出任意长度且均匀分布的比特串的扩展输出函数,则$y \sim S := \mathrm{Sam}(x)$表示Sam在输入$x$的情况下产生一个均匀分布在集合$S$上的值$y$.AKCN-MLWE 中使用 SHA-3[15]中的 SHAKE-128 与 SHAKE-256 作为可扩展输出函数,分别用于矩阵生成与噪音采样.

二项分布采样:AKCN-MLWE 的私钥和错误向量由中心二项分布β_η生成,β_η定义为:取样$\{(a_i,b_i)\}_{i=1}^{\eta} \leftarrow (\{0,1\}^2)^\eta$,并输出$\sum_{i=1}^{\eta}(a_i-b_i)$.其中$\eta$是某些正整数.如果某个元素$v \in R$或向量$\mathbf{v} \in R^k$是通过中心二项分布生成的,则表明$v$(或$\mathbf{v}$)的每项系数是通过中心二项分布$\beta_\eta$(或$\beta_\eta^k$)生成的.

数论变换(NTT):定义 NTT 算法为前向数论变换,即将多项式$a=\sum_{i=0}^{n-1} a_i X_i \in R_q$转换成 NTT 域上的多项式$\hat{a}$.相应地,定义 NTT^{-1} 算法逆向数论变换,NTT^{-1}则是将 NTT 形式的$\hat{a}$转换成环R_q的多项式a,数论变换有如下性质:$a=\mathrm{NTT}^{-1}(\mathrm{NTT}(a))$,并且$c=ab=NTT^{-1}(NTT(a)\circ NTT(b))$.

压缩与解压缩:定义函数$\mathrm{Compress}_q(x,d)$,输入$x \in \mathbb{Z}_q$并输出一个在$[0,2^d)$范围内的整数,其中$d<\lfloor log_2(q)\rfloor$.另外再定义一个$\mathrm{Decompress}_q$函数使得$x' = \mathrm{Decompress}_q(\mathrm{Compress}_q(x,d),d)$,$x'$是一个很接近$x$的元素.在 AKCN-MLWE 中,压缩函数与解压缩函数具体为

$$x'' = \mathrm{Compress}_q(x,d) = \lfloor x\cdot 2^d/q \rceil$$
$$x' = \mathrm{Decompress}_q(x'',d) = \lfloor x''\cdot q/2^d \rceil$$

定义压缩与解压缩函数的目的是可以舍弃加密过程中公钥和密文的一些对解密的正确性影响不大的低位比特,从而降低公钥和密文的存储空间.

非对称密钥共识机制 AKC(Asymmetric Key Consensus):一个 AKC 机制包含一对多项式时间算法算法,Con 和 Rec,用于会话双方达成密钥共识.定义函数$\mathrm{Con}(\sigma_1,k_1,params)$,输入$\sigma_1 \in \mathbb{Z}_q, k_1 \in \mathbb{Z}_m$,以及系统参数$params=(q,m,g)$,输出提示信息$v \in \mathbb{Z}_g$.函数$\mathrm{Rec}(\sigma_2,v,params)$输入$\sigma_2 \in \mathbb{Z}_q$和$v$,输出$k_2 \in \mathbb{Z}_m$.在 AKCN-MLWE 中,共识算法的具体形式为

$$\begin{aligned} v &= \mathrm{Con}(\sigma_1,k_1,params) \\ &= \lfloor g(\sigma_1+\lfloor k_1 q+m\rfloor)/q\rfloor \bmod m \\ k_2 &= \mathrm{Rec}(\sigma_2,v,params) \\ &= \lfloor m(v/g-\sigma_1/q)\rceil \bmod m \end{aligned}$$

正确且安全的共识机制保证了当$|\sigma_1-\sigma_2| \le d$时有$k_1=k_2$,且当$\sigma_1$在$\mathbb{Z}_q$均匀分布时有$v$与$k_1$的分布独立.

3.2 AKCN-MLWE

AKCN-MLWE[2]方案是基于非对称密钥共识和 MLWE 困难性问题的密钥封装方案的模块化通用构造.AKCN-MLWE 的核心算法主要包含两部分,IND-CPA 安全的公钥加密(AKCN-MLWE-PKE=(KeyGen, Enc, Dec))算法,和 IND-CCA2 安全的密钥封装机制(AKCN-MLWE-KEM=(KeyGen, Encaps, Decaps)),其中 AKCN-MLWE 中的密钥封装机制是通过对公钥加密算法进行 FO 变换(Fujisaki-Okamoto

transformation)[16]构建而成.由于本文着重介绍AKCN-MLWE 的底层运算的快速实现,因此本文只简要介绍 AKCN-MLWE 中 IND-CPA 安全的公钥加密算法,进而展开对 AKCN-MLWE 底层运算的快速实现.有关AKCN-MLWE中的密钥封装机制,读者可参考文献[2]详细了解.

AKCN-MLWE 是建立在多项式环 $R_q = Z_q[X]/(X^n+1)$上的,其算法参数的选取使得其底层的多项式乘法运算可以使用 NTT 来进行快速实现.不过与 RLWE 不同的是,为了在安全和性能之间作一个平衡,基于 MLWE[9]困难假设的设计方案在多项式环结构上引进了一个很小的维度 k (通常为2,3,4),并降低 RLWE 中的维度 n 来保证运行效率与 RLWE 相当.另外基于 MLWE 的方案相对于 RLWE 还具有额外的灵活性,在设计不同安全等级的方案时,只需要调整维度 k 即可,环 R_q 中的基本运算可以被不同安全强度的方案复用.而在基于 RLWE 的方案中,不同安全等级的方案则需要更改环 R_q 并且需要重复实现环的基本运算.AKCN-MLWE 引入的 k 维矩阵为 $\mathbf{A} \in R_q^{k\times k}$.在此基础上,MLWE 的数学困难问题是指针对两元组 $(\mathbf{A},\mathbf{t})$,其中 $\mathbf{A} \in R_q^{k\times k}$, $\mathbf{t} \in R_q^k$,区分 $\mathbf{t}$ 是从 R_q^k 中随机选取的还是通过特定的 $\mathbf{s},\mathbf{e} \in R_q^k$ 通过计算 $\mathbf{As}+\mathbf{e}$ 得到的是困难的.AKCN-MLWE-768 有两组推荐参数,如表 1 所示.其中,$|K|$ 表示密钥封装过程中产生的共享密钥 K 的长度,共享密钥 K 可作为对称加密的密钥对通信双方传输的数据进行加密传输,以保障数据机密性.n 为多项式环 R_q 中多项式元素的维度,q 为模数.

表 1　AKCN-MLWE 算法两组推荐参数

算法	$\|K\|$	n	q	η	g
AKCN-MLWE-768-1	256	256	7681	2	2^3
AKCN-MLWE-768-2	256	256	7681	2	2^4
算法	t_1	t_2	l	$csec$	pqsec
AKCN-MLWE-768-1	3	3	3	163	148
AKCN-MLWE-768-2	0	4	3	163	148
算法	err	$pk(B)$	$c(B)$	$bw.(B)$	
AKCN-MLWE-768-1	$2^{-166.4}$	992	1056	2048	
AKCN-MLWE-768-2	2^{-128}	1280	992	2272	

其次,AKCN-MLWE 在设计时,为了便于比较和适配性,采用了与 Frodo-KEM 相同的噪音分布,AKCN-MLWE 的私钥和错误向量由中心二项分布 β_η 生成,表 1 中的 η 表示噪音分布参数;g 用于表示非对称密钥共识机制中 v 系数的取值空间(见 3.1 节);另外,由于公钥和密文中的一些低位比特对解密概率的正确性没有影响,因此 AKCN-MLWE 为了减少公钥和密文的大小,减少传输过程的带宽,设置两个压缩系数 t_1 与 t_2,分别表示公钥和密文压缩的比特数.l 为数组维数,$csec$ 与 pqsec 分别表示对应参数的 AKCN-MLWE 具有的传统与量子安全强度,err 为错误率,$pk(B)$、$c(B)$、$bw.(B)$ 分别表示公钥、密文与 KEM 算法过程中传输数据 (pk,c) 的带宽大小,单位为字节.

表中两组参数在公钥和密文长度上有不同的表现,每组参数都相对 128 传统安全强度有充足的安全冗余.算法推荐使用 AKCN-MLWE-768-1 这组参数,本文也只针对此组参数进行实现.

算法 1. AKCN-MLWE.PKE.KeyGen 密钥生成

1: $(\rho,\sigma) \leftarrow \{0,1\}^{256}$
2: $\hat{\mathbf{A}} \sim R_q^{k\times k} := \mathrm{Parse}(\mathrm{Sam}(\rho))$
　　▷生成 NTT 域的矩阵 $\hat{\mathbf{A}} \in R_q^{k\times k}$
3: $(\mathbf{s},\mathbf{e}) \sim \beta_\eta^k \times \beta_\eta^k := \mathrm{CBD}_\eta(\sigma)$
　　▷从 β_η 中产生私钥和噪音向量 $\mathbf{s},\mathbf{e} \in R_q^k$
4: $\hat{\mathbf{s}} := \mathrm{NTT}(\mathbf{s})$
5: $\mathbf{t} := \mathrm{NTT}^{-1}(\hat{\mathbf{A}} \circ \hat{\mathbf{s}}) + \mathbf{e}$
6: $pk := (\mathrm{Compress}_q(\mathbf{t}, d_v)||\rho)$ ▷ $pk := \mathbf{As}+\mathbf{e}$
7: $sk := \hat{\mathbf{s}} \bmod q$ ▷ $sk := \mathbf{s}$
8: **RETURN**(pk, sk)

AKCN-MLWE-PKE 的密钥生成如算法 1 所示,在 AKCN-MLWE 密钥生成算法中,最主要的运算是计算 $\mathbf{As}+\mathbf{e}$,为了使用 NTT 对这个运算进行加速,AKCN-MLWE 首先通过Sam和Parse算法,以 ρ 为种子生成在NTT域的矩阵 $\hat{\mathbf{A}}$,然后将其与 $\mathbf{s}$ 的NTT形式 $\hat{\mathbf{s}}$ 进行点乘,最后将NTT域的向量结果转换成正常的多项式形式并进行后续计算.注意私钥 sk 是以 NTT 形式进行存储的,因为后续需要利用私钥进行多项式乘法计算,这样可以避免重复进行数论变换,提高实现效率.而公钥则需要转换成正常的多项式形式进行存储,这样可以利用Compress函数缩小公钥的存储空间,另外利用Compress和Decompress函数还可用于在加密和解密过程中执行错误纠正.

算法 2. CPA-安全的 AKCN-MLWE.PKE.Enc 加密算法

输入:公钥 $pk=(t,\rho)$,明文 m,随机数 r

输出:密文c

1: $\mathbf{Y} \coloneqq \mathrm{Decompress}(\mathbf{t}, d_v) \triangleright \mathbf{Y} \coloneqq \mathbf{t} \cdot q/2^{d_v}$

2: $\widehat{\mathbf{A}} \sim R_q^{k\times k} \coloneqq \mathrm{Parse}(\mathrm{Sam}(\rho))$ ▷生成 NTT 域的矩阵$\widehat{\mathbf{A}} \in R_q^{k\times k}$

3: $(\mathbf{r}, \mathbf{e_1}, e_2) \sim \beta_\eta^k \times \beta_\eta^k \times \beta_\eta \coloneqq \mathrm{CBD}_\eta(r)$ ▷从β_η中产生噪音向量$\mathbf{s}, \mathbf{e_1} \in R_q^k, e_2 \in R_q$

4: $\hat{\mathbf{r}} \coloneqq \mathrm{NTT}(\mathbf{r})$

5: $\widehat{\mathbf{Y}} \coloneqq \mathrm{NTT}(\mathbf{Y})$

6: $\mathbf{u} \coloneqq \mathrm{NTT}^{-1}(\widehat{\mathbf{A}^\mathbf{T}} \circ \hat{\mathbf{r}}) + \mathbf{e_1} \triangleright \mathbf{u} \coloneqq \mathbf{A^T r} + \mathbf{e_1}$

7: $v \coloneqq \mathrm{NTT}^{-1}(\widehat{\mathbf{Y}} \circ \hat{\mathbf{r}}) + e_2 \triangleright v \coloneqq \mathbf{Y^T r} + e_2$

8: $c_1 \coloneqq \mathrm{Compress_q}(\mathbf{u}, d_u)$

9: $c_2 \coloneqq \mathrm{Con}(v, m, params)$

10: **RETURN** $c = (c_1||c_2)$

算法 3. CPA-安全的 AKCN-MLWE.PKE.Dec 解密算法

输入:私钥$sk = s$,密文$c = (\mathbf{u}, v)$

输出:明文m

1: $\mathbf{u} \coloneqq \mathrm{Decompress_q}(\mathbf{u}, d_u)$

2: $\mathbf{\Sigma} \coloneqq s^\mathrm{T} \circ \mathrm{NTT}(\mathbf{u})$

3: $m \coloneqq \mathrm{Rec}(\mathbf{\Sigma}, v, params)$

4: **RETURN**m

AKCN-MLWE 的公钥加密和解密算法如算法 2,算法 3 所示.与密钥生成算法相同,加密算法中矩阵$\widehat{\mathbf{A}}$的生成是使用相同的种子ρ通过Sam和Parse函数生成的.由于种子ρ是作为公钥的一部分被公开的,这可以保证加密时使用的矩阵$\widehat{\mathbf{A}}$与公钥生成时的矩阵一致.进而,使用随机数r作为种子从β_η中产生错误向量$(\mathbf{r}, \mathbf{e_1}, e_2)$,进而再计算密文$\mathbf{u}, v$.

3.3 数论变换

数论变换(NTT, Number Theoretic Transform)是环R_q中多项式乘法运算的一个快速实现算法,它是离散傅立叶变换(DFT, Discrete Fourier Transform)的一个变种.不过由于 DFT 是在复数域上的转换,在计算过程中需要用到复数或浮点数并且会存在舍入问题,这使得 DFT 不适用于计算准确度要求非常高的密码算法中,而 NTT 作为整数域上的 DFT 变种,更适用于密码算法的快速实现.

对于多项式环R_q上两个n阶多项式$a = \sum_{i=0}^{n-1} a_i X^i, b = \sum_{i=0}^{n-1} b_i X^i$.利用 NTT 计算多项式乘法$c = ab$主要包含以下三个过程:

1)NTT 前向变换(NTT):将两个多项式a, b分别通过下述公式转换到 NTT 域,其中q是素数,w^{ij}表示$\mathbb{Z}_q$中单位元的第i个原根(亦称旋转因子),NTT 域上的元素a, b分别用$\hat{a}, \hat{b}$表示.当需要将矩阵或向量转换到 NTT 域时,表示对矩阵或向量中的每个多项式分别进行 NTT.NTT 输入正常顺序的多项式a,然后输出位反转顺序的元素$\hat{a}$.

$$\hat{a} = \mathrm{NTT}(a) = \sum_{j=0,i=0}^{n-1} a_j w^{ij} \bmod q$$

2)点乘(∘):转换成NTT域的元素$\hat{a}, \hat{b}$进行点乘运算,即$\widehat{c_i} = \widehat{a_i} \circ \widehat{b_i}, i \in [0, n)$.只需要$n$个乘法运算即可,而且每个乘法运算是相互独立的,因此可以利用 AVX2 等并行技术进行加速.

3)NTT 逆向变换(NTT^{-1}):利用下述公式将多项式乘法的乘积$\hat{c}$转换成正常形式c,NTT^{-1}过程与 NTT 类似,但需要使用逆旋转因子w^{-1}.其中,$\gamma = \sqrt{w}$.

$$c = \mathrm{NTT}^{-1}(\hat{c}), c_i = n^{-1}\gamma^{-i} \sum_{j=0}^{n-1} \widehat{c_j} w^{-ij} \bmod q$$

上述 NTT 和 NTT^{-1}的算法复杂度均为$O(n^2)$,利用 CT-FFT (Cooley-Tukey Fast Fourier Transform)可以通过分而治之的思想将 DFT(或 NTT)的时间复杂度进一步降低,CT-FFT 的主要流程是将n阶多项式通过二分法分解成n_1个阶为n_2的多项式$(n = n_1 \times n_2)$,然后对每个阶为n_2的多项式再进一步地进行 DFT(或 NTT)变换,这个过程也可以递归进行直到将每个多项式的阶降低到$n_2 = 1$.CT-FFT 二分法$(n_1 = 2)$是一个比较常用的快速实现方案,利用 CT-FFT 二分法,NTT 的时间复杂度可降低到$O(n \cdot \log n)$.

一个完整的多项式乘法使用 CT-FFT 二分算法的计算流程为$c = ab = \mathrm{NTT}^{-1}(\mathrm{NTT}(a) \circ \mathrm{NTT}(b))$.通过该算法完成一个完整的多项式乘法的时间复杂度为$2O(n \cdot \log n) + O(n) = O(n \log n)$,而且该算法可以充分利用 AVX2 的并行性进行加速,相对于传统的需要$O(n^2)$时间复杂度的教科书乘法,使用 NTT 的多项式乘法运行效率可以得到极大的提升.此外在 AKCN-MLWE 中,部分多项式乘法可以通过保存 NTT 域元素来减少 NTT 前向变换的次数,以进一步提高运行效率.例如矩阵$\mathbf{A}$在生成过程中直接假设随机生成的数是 NTT 域上的元素,只需要进行简单的位反转处理即可,这是由于在R_q上均匀分布的元素经过 NTT 也是均匀分布的.另外私钥sk也是以

NTT 形式进行存储,可以有效降低 NTT 前向变换的次数.

算法 4. 前向 NTT 算法

输入:正常形式多项式a,预计算的旋转因子w

输出:NTT 域元素$\hat{a}$

```
FOR NP=n/2; NP>0; NP/=2 DO
    i=0; j=0; k=0;
    FOR i =0; i<n; i=j+NP DO
        FOR j=0; j<i+NP; j++ DO
            t = (w[k] × a[j + NP]) mod q;
            a[j + NP] = (a[j] − t) mod q;
            a[j] = (a[j] + t) mod q;
        END FOR
        k++;
    END FOR
END FOR
```

AKCN-MLWE 中的前向 NTT 和逆向 NTT 算法如算法 4,算法 5 所示,这两个算法流程是类似的,都由 3 个主循环构成,不过内循环的蝴蝶变换有所不同,分别采用的 CT 蝴蝶变换(Cooley-Tuckey Butterflies)和 GS 蝴蝶变换(Gentleman-Sande Butterflies).这两种蝴蝶变换的主要区别在于 CT 变换输入正常顺序的多项式,输出位反转顺序的 NTT 域元素;而 GS 变换则输入位反转顺序的 NTT 域元素,输出正常顺序的多项式.在前向 NTT 和逆向 NTT 过程中分别使用 CT 和 GS 变换可以避免昂贵的位反转操作,以提高运行效率.

算法 5. 逆向 NTT 算法

输入:NTT 域的元素$\hat{a}$,预计算的旋转因子w^{-1}

输出:正常形式的多项式a

```
FOR NP=1; NP<n; NP*=2 DO
    i=0; j=0; k=0;
    FOR i=0; i<n; i=j+NP DO
        FOR j=0; j<i+NP; j++ DO
            t = a[j];
            a[j] = (t + a[j + NP]) mod q;
            a[j + NP] = (w^{-1}[k]t − a[j + NP]) mod q;
        END FOR
        k++;
    END FOR
END FOR
```

3.4 AVX2指令集

AVX2 是英特尔处理器中的一种单指令多数据(SIMD)指令集,单条 AVX2 指令可以处理 256 位的寄存器数据,单条指令集可以对 4 组 64 位数据,8 组 32 位数据,甚至于 16 组 16 位数据同时进行运算.此外,AVX2 提供的指令集种类也非常齐全,包括算术运算,逻辑运算,排列,广播,混合等,可以满足数据处理的各种需求.AVX2 强大的并行性及齐全的指令集使得其非常适用于图像处理,科学计算等领域.在格密码算法中,由于 NTT 具有比较好的可并行性,而且其素数q通常在 16 比特以内,因此 AVX2 非常适用于格密码的优化实现.AVX2 中提供的对 16 组 16 位数据进行乘法算术运算的指令 vpmulhw 和 vpmullw 可以将 16 位整数的乘积分成高低半部分分别进行计算,在 Haswell 和 Skylake 处理器中,这两条指令的时延均为 5 个时钟周期,因此计算 16 组 16 位数据的乘法运算只需要 10 个时钟周期.在蒙哥马利约减算法中,由于部分运算只需要保留乘积的高或低半部分,部分乘法指令可以被省略,整体运行效率会更快.由于 AKCN-MLWE 中的模数q小于 16 比特,因此我们主要使用 AVX2 指令集中以 16 比特存储的指令,这样同一条指令可以处理更多组数据.

4 AKCN-MLWE 算法 AVX2 高效实现方案设计

本文提出的针对 AKCN-MLWE 算法的优化方案主要包括以下关键点:

1)多项式乘法:将 NTT 的最后一层转换为线性多项式向量点乘并使用 Karatsuba 算法进行加速计算,使用更小的预计算表.

2)模约减:结合了高效的 Barrett 约减和蒙哥马利约减两种约减算法,同时通过精确计算算法中每一步的取值边界,使用延迟约减降低了取模次数.

3)多项式运算:使用 AVX2 对多项式加法、减法,多项式序列化、反序列化和多项式编码、解码进行并行计算或处理.

4.1 改进的多项式乘法

本文采用 Kyber 在 NIST 第二轮中的思路,删减 NTT 的最后一层,这样可以提升前向 NTT 和逆向 NTT 的性能,而且可以降低预计算表大小.代价是点

乘阶段由两个n维向量点乘变为两个n维线性多项式向量点乘.

4.1.1 线性多项式向量点乘

对于本文使用的 NTT 方法,在对两个n维多项式分别进行前向NTT之后得到两个n维线性多项式向量,点乘即转换为两个线性多项式的乘法.从数学角度考虑,教科书式的n维线性多项式乘法一共需要$5n$次乘法,$2n$次加法/减法;而 Karatsuba 算法需要$4n$次乘法,$5n$次加法/减法.从效率角度考虑两种实现方案,Karatsuba 算法减少了乘法指令,引入了更多的加法/减法指令,实际性能基本与教科书式算法相当.若特定 CPU 上的乘法指令延迟较低、吞吐较大,那么教科书式算法拥有更好的性能,反之 Karatsuba 算法更快.

本文同时给出教科书式和 Karatsuba 算法的线性多项式乘法的AVX2设计方案,分别见算法6和算法 7.

线性多项式乘法定义如下:已知删减一层后的 NTT 域上的两个向量$\widehat{\boldsymbol{a}}$和$\widehat{\boldsymbol{b}}$,此形式下的向量点乘即计算$\widehat{\boldsymbol{a}}\cdot\widehat{\boldsymbol{b}}\ \boldsymbol{mod}\ (\boldsymbol{x^2}\pm\zeta)$.记$\widehat{\boldsymbol{a}}$和$\widehat{\boldsymbol{b}}$中对应位置的元素分别为$a+bx$和$c+dx$,记结果中对应位置的元素为$e+fx$.教科书式的做法为交叉相乘,即,

$$e+fx=\big((a+bx)\cdot(c+dx)\big)mod(x^2\pm\zeta)$$
$$=(ac\pm\zeta bd)+(ad+bc)x.$$

单个线性多项式乘法需要 5 次乘法和 2 次加法/减法.Karatsuba 算法的思想为,通过额外计算a + b,c + d,以及$(a+b)\cdot(c+d)$,从而免去计算ad和bc.

$$e+fx=\big((a+bx)\cdot(c+dx)\big)mod(x^2\pm\zeta)$$
$$=(ac\pm\zeta bd)+\big((a+b)(c+d)-ac-bd\big)x.$$

单个线性多项式乘法需要 4 次乘法和 5 次加法/减法.

算法 6. AVX2 教科书式线性多项式乘法

输入:向量化的两个线性多项式$a+bx$和$c+dx$

输出:向量化的多项式乘积$e+fx=\big((a+bx)\cdot(c+dx)\big)\mathrm{mod}(x^2\pm r)$

1: vmovdqa $a,\ b,\ c,\ d$ ▷载入
2: vpmul{l|h}w $\{ac,\ ad,\ bc,\ bd\}_{\{hi,lo\}}\leftarrow\{ac,\ ad,\ bc,\ bd\}$ ▷乘法
3: vpmul{l|h}w, vpsubw $bd'\leftarrow bd_{\{hi,lo\}}$ ▷约减
4: vpmul{l|h}w $rbd_{\{hi,lo\}}\leftarrow r\cdot bd$ ▷乘法
5: vpunpck{l|h}wd $\{ac,\ ad,\ bc,\ rbd\}\leftarrow\{ac,\ ad,\ bc,\ rbd\}_{\{hi,lo\}}$ ▷解包
6: vpaddd,vpsubd $\{e,\ f\}\leftarrow\{ac\pm rbd,\ ad+bc\}$ ▷加法或减法

算法 7. AVX2 Karatsuba 线性多项式乘法

输入:向量化的两个线性多项式$a+bx$和$c+dx$

输出:向量化的多项式乘积$e+fx=\big((a+bx)\cdot(c+dx)\big)\mathrm{mod}(x^2\pm r)$

1: vmovdqa $\{a,\ b,\ c,\ d\}$ ▷载入
2: vpaddw $\{t_1,\ t_2\}\leftarrow\{a+b,\ c+d\}$ ▷加法
3: vpmul{l|h}w $\{ac,\ bd,\ m\}_{\{hi,lo\}}\leftarrow\{a\cdot c,\ b\cdot d,\ t_1\cdot t_2\}$ ▷乘法
4: vpmul{l|h}w, vpsubw $bd'\leftarrow bd_{\{hi,lo\}}$ ▷约减
5: vpmul{l|h}w $rbd_{\{hi,lo\}}\leftarrow r\cdot bd$ ▷乘法
6: vpunpck{l|h}wd $\{ac,\ bd,\ rbd,\ m\}\leftarrow\{ac,\ bd,\ rbd,\ m\}_{\{hi,lo\}}$ ▷解包
7: vpaddd $n\leftarrow ac+bd$ ▷加法
8: vpaddd, vpsubd $\{e,f\}\leftarrow\{ac\pm rbd,m-n\}$ ▷加法或减法

Karatsuba 算法相较于教科书式方法可以节约 2 条 AVX2 乘法指令,分别为 vpmullw 和 vpmulhw,代价是增加 2 条加法指令 vpaddw 和两条减法指令 vpsubd.两种方法都可高度利用流水线和数据并行.

本文在实验中采用算法 7 中的 Karatsuba 算法进行线性多项式向量点乘.

4.1.2 预计算表

采用预计算表可以通过增加一定的空间占用来降低运算时间.本文中主要预计算了ζ、ζ^{-1}、ζq^{-1}以及$\zeta^{-1}q^{-1}$等来加速 NTT 各个阶段的计算.在构造用于 AVX2 实现的预计算表时,为了省去不必要的广播指令,针对每一层的预计算值都重复了一定的次数,使得在并行载入时可以一次全部载入.

本文由于将 NTT 的最后一层转换为线性多项式进行加速计算,因此预计算量可减半,一共仅使用1584字节.

4.2 快速模约减

由于 C 语言中的取模运算(%)中使用除法,效率较低,Barrett 约减算法[17]将除法转换为乘法,从而提升效率.蒙哥马利约减算法[18]是另一种高效的模约减算法,但其会转换到 MONT 域,在约减完还需要转换回到正常域.值得一提的是,NTT 同样也需要转换到NTT域,且可与MONT域的转换相结合,从实现角度可以做到完全省去 MONT 域的转换开销.

在参考实现中,前向 NTT 和逆向 NTT 中每一层都进行了 Barrett 约减,但考虑到存储结构的限制和约减算法的输入输出范围,其中特定层完全可以不进行约减.

4.2.1 Barrett 约减

计算$r = a \bmod q$,其中a为 16 位有符号整数,$-2^{15} \le a < 2^{15}$,q为模数,r为余数,$0 \le r < q$.标准的约减需要一次整数除法,但是可以通过倒数形式表示为乘法,

$$r = a \bmod q = a - q \cdot \left\lfloor \frac{a}{q} \right\rfloor = a - q \cdot \lfloor a \cdot q^{-1} \rfloor$$

进一步地,应用改进的 Barrett 约减算法[12, 17],见算法 8.

算法 8. 改进的 Barrett 约减算法

输入:16 位有符号整数a满足$-\beta/2 \le a < \beta/2$,模数q满足$q < \beta/2$

输出:$r \equiv a(\bmod q)$,其中$0 \le r \le q$

1: $v = \left\lfloor \frac{2^{\lfloor log(q) \rfloor - 1}\beta}{q} \right\rfloor$

2: $t = \left\lfloor \frac{av}{2^{\lfloor log(q) \rfloor - 1}\beta} \right\rfloor$

3: $r = a - (tq \bmod \beta)$

算法 9. 改进的 Barrett 约减算法的 AVX2 实现

输入:向量化的 16 位有符号整数a, 常数模数 q 以及预计算的$v = \left\lfloor \frac{2^{\lfloor log(q) \rfloor - 1}\beta}{q} \right\rfloor$和$x = \lfloor \log(q) \rfloor - 1$

输出:向量化的约减后的值,记为r

1: vpmulhw $t \leftarrow a \cdot v$ ▷计算$a \cdot v$,取高 16 位

2: vpsraw $t \leftarrow t \gg x$ ▷算术右移x位

3: vpmullw $t \leftarrow t \cdot q$ ▷计算$t \cdot q$,取低 16 位

4: vpsubw $r \leftarrow a - t$ ▷计算$a - t$,得到结果

其中v由常量计算得到,可以进行预计算.令$\beta = 2^{16}$,保证a的上下界在$\pm\beta/2$之间,代入算法得

$$t = av \gg \left(\lfloor \log(q) \rfloor - 1 + \log(\beta)\right),$$

此时计算t仅需要求得a与v的乘积高字,接着计算$tq \bmod \beta$等价于计算t与q的乘积低字.如算法 9,使用 AVX2 指令集中的并行单字乘法,可以用 Barrett 约减算法并行计算 16 个约减且仅需 4 条 AVX2 指令.值得注意的是,当此 Barrett 约减算法的输入为$-q$的倍数时,约减后的结果为q而不是 0.

4.2.2 蒙哥马利约减

蒙哥马利约减算法[18]首先选择一个基数β,使得β与q互素且$\beta > q$,为了计算高效一般取 2 的整数倍.当满足$0 \le a < \beta q$时,将计算$r = a \bmod q$转换为先计算$r' = a\beta^{-1} \bmod q$,再计算$r = r'\beta \bmod q$.计算r'用到的算法即为蒙哥马利约减算法.算法最后判断r'若在$[q, 2q)$之间则减q,从而保证最终r'范围为$[0, q)$.

本文使用经过修改的有符号蒙哥马利约减算法[12],见算法 10.与原始的蒙哥马利约减算法不同的是,算法可接受的输入范围调整为$[-\beta q/2, \beta q/2)$,同时省略了原算法最后进行的判断减q操作,从而得到的r'的范围为$(-q, q)$.

算法 10. 改进的蒙哥马利约减算法

输入:32 位有符号整数a满足$-\beta q/2 \le a < \beta q/2$

输出:16 位有符号整数r'满足$-q < r' < q$

1: $m = aq^{-1} mod^{\pm} \beta$ ▷ $-\beta/2 \le m < \beta/2$

2: $t = \lfloor m \cdot q/\beta \rfloor$

3: $r' = \lfloor a/\beta \rfloor - t$

算法 11. 改进的蒙哥马利约减算法 AVX2 实现

输入:向量化的 32 位有符号整数a,记为a_{hi}和a_{lo}

输出:向量化的 16 位有符号整数r'

1: vpmullw $m \leftarrow a_{lo} \cdot q^{-1}$ ▷ $m = (a_{lo} \cdot q^{-1}) \bmod^{\pm} \beta$

2: vpmulhw $t \leftarrow m \cdot q$ ▷ t = $\lfloor m \cdot q/\beta \rfloor$

3: vpsubw $r' \leftarrow a_{hi} - t$ ▷ $r' = a_{hi} - t$

算法 11 描述了 AVX2 实现的蒙哥马利约减算法.算法的输入是两个 256 位的寄存器,里面分别存着 16 个 32 位有符号数的高 16 位(hi)和低 16 位(lo).这里选取其中一个数a,首先用 vpmullw 指令计算$a_{lo} \cdot q^{-1}$的低 16 位记为m,对应算法 10 中的第一步.接着使用 vpmulhw 指令计算$m \cdot q$的高 16 位,即$\lfloor m \cdot q/\beta \rfloor$,得到$t$.最后使用 vpsubw 指令计算$a_{hi} - t$得到$r'$.

4.2.3 延迟约减

为了减少不必要的约减,从而提升效率,本文针对前向 NTT、点乘和逆向 NTT,根据蒙哥马利约减算法和 Barrett 约减算法的输入输出范围,精确计算出 NTT 每一层的输入输出范围,最大化地减少约减次数.

前向 NTT 中,第零层的多项式系数输入范围为$[0, q)$,由于蒙哥马利约减算法的输出范围为$(-q, q)$,

所以经过 CT 蝴蝶变换后的系数输出范围为$(-q,2q)$,此时并不需要 Barrett 约减,下一层可以接受该范围的输入.同理,第一层的输入范围为$(-q,2q)$,输出范围为$(-2q,3q)$;第二层的输入范围为$(-2q,3q)$,输出范围为$(-3q,4q)$.此时下一层 CT 蝴蝶变换的输入范围将是$(-4q,5q)$,代入$q=7681$已经从 16 位有符号数溢出.因此第二层蝴蝶变换结束后需要对所有系数进行 Barrett 约减,保证下一层的计算正确性.以此类推,只需要在第五层结束后再进行一次 Barrett 约减即可.考虑到接下来的点乘中无法接受$(-q,2q)$的输入,第六层结束后仍然需要做 Barrett 约减.因此整个前向 NTT 仅需在第二、五、六层进行 Barrett 约减.前向 NTT 结束后多项式系数范围为$[0,q]$.

点乘中,输出范围来自前向 NTT 的输出范围,为$[0,q]$,由于需要做三次点乘后将三个多项式求和,本文将第一个和第二个多项式求和后再进行蒙哥马利约减,然后将第三个多项式单独做蒙哥马利约减,再与之前的约减结果求和,因此输出范围为$(-2q,2q)$.

逆向 NTT 中,第零层的输入范围由点乘的输出范围决定,为$(-2q,2q)$,经过 GS 蝴蝶变换后输出范围为$(-4q,4q)$,此时需要 Barrett 约减.以此类推,整个逆向 NTT 中仅需在第零、二、四层进行 Barrett 约减.由于第六层结束后进行了蒙哥马利约减,所以逆向 NTT 结束后多项式系数范围为$(-q,q)$.

4.3 多项式运算

AKCN-MLWE 中需要进行的多项式运算包括:多项式乘法、多项式加减法、Con、Rec、多项式序列化与反序列化以及对消息的编码与解码.

本文在多项式乘法优化的基础上,对多项式加法与减法、压缩与解压缩、序列化与反序列化以及消息的编码与解码进行了 AVX2 并行优化.其中压缩解压缩与编码解码是 Con 与 Rec 中的核心步骤.

多项式加减法、消息的编码解码采用了 Kyber[13]中的并行优化方法.多项式序列化与反序列化的并行采用了 NTTRU[14]中的方法,即每隔 16 个系数进行序列化,以满足 AVX2 指令集的数据对齐要求,从而最大程度地并行运算.

在多项式压缩中包含了除q四舍五入运算,由于 AVX2 中没有除法指令,对其进行并行实现难度较大.本文提炼 Barrett 约减[17]中的优化思路设计了一种新型的方法对此运算进行了转化,即用乘法、加法与移位,来完成高效且正确的压缩运算.

令$m_{exact}=2^k/q$,则a除q的结果可表示为

$$\left\lfloor\frac{a}{q}\right\rfloor=\left\lfloor\frac{a}{q}\cdot\frac{2^k}{2^k}\right\rfloor=\left\lfloor\frac{2^k}{q}\cdot\frac{a}{2^k}\right\rfloor=\left\lfloor m_{exact}\cdot\frac{a}{2^k}\right\rfloor$$

由于m_{exact}非整数,所以取$m=\lceil m_{exact}\rceil=\frac{2^k+e}{q}$来近似表示,其中$0<e<q$.从而得到

$$\left\lfloor\frac{a}{q}\right\rfloor=\left\lfloor m\cdot\frac{a}{2^k}\right\rfloor=\left\lfloor\frac{a}{q}+\frac{ea}{2^kq}\right\rfloor$$

为了得到正确结果,必须选择合适的k.k的取值应满足

$$\frac{ea}{2^kq}=\frac{e}{q}\cdot\frac{a}{2^k}<\frac{1}{q}$$

因为$0<e<q$,所以有$k>log_2 aq$.

因此$\left\lfloor\frac{a}{q}\right\rfloor=(a\cdot m)\gg k$,其中$k\geq\lceil\log_2 aq\rceil$,$m=\left\lceil\frac{2^k}{q}\right\rceil$,$k$的取值下界由a的取值上界决定.将此方法代入压缩与解压缩公式,并进行进一步转化,得到如下计算公式:

$$\begin{aligned}x''&=\mathrm{Compress}_q(x,d)=\lfloor x\cdot 2^d/q\rceil\\&=\lfloor((x\ll d)+\lfloor q/2\rfloor)/q\rfloor\\&=\left(((x\ll d)+\lfloor q/2\rfloor)\cdot m\right)\gg k\end{aligned}$$

$$\begin{aligned}x'&=\mathrm{Decompress}_q(x'',d)=\lfloor x''\cdot q/2^d\rceil\\&=(x''\cdot q+2^{d-1})\gg d.\end{aligned}$$

算法 12 中描述了将多项式系数压缩到 3 比特以及从 3 比特解压缩的算法.代入 AKCN-MLWE 算法的实例化参数,模数$q=7681$.在压缩至 3 比特的情况下,将$d=3$代入求得被除数的取值上界为$(q\ll d)+\lfloor q/2\rfloor=65288$,由此计算出$k$和$m$的最小取值分别为$k=29$和$m=69896$,可以进一步化简为$m=8737$和$k=26$.代入上述公式得到最终的实现公式,

$$\begin{aligned}x''&=\mathrm{Compress}_{7681}(x,3)\\&=\left(((x\ll 3)+3840)\cdot 8737\right)\\&\gg 29\end{aligned}$$

$$\begin{aligned}x'&=\mathrm{Decompress}_{7681}(x'',3)=(x''\cdot 7681+4)\\&\gg 3.\end{aligned}$$

由于 AVX2 中的 16 位乘法分为两条指令,分别得到乘积的高 16 位和低 16 位.因此,考虑到最后一步计算为右移 29 位,只需使用 vpmulhuw 指令得到乘积的高 16 位.

算法 12.3 比特压缩与解压缩的 AVX2 实现

输入: 16 个多项式系数组成的a

输出:每个系数均压缩至 3 比特后的a

1: vpsllw $a[i] \leftarrow a[i] \ll 3$

2: vpaddw $a[i] \leftarrow a[i] + 3840$

3: vpmulhuw $a[i] \leftarrow a[i] \times 8737$

4: vpsllw $a[i] \leftarrow a[i] \ll 3$

5: vpsrlw $a[i] \leftarrow a[i] \gg 13$

输入:16 个系数均为 3 比特的a

输出:解压缩后的a

1: vpmullw $a[i] \leftarrow a[i] \times q$

2: vpaddw $a[i] \leftarrow a[i] + 4$

3: vpsrlw $a[i] \leftarrow a[i] \ll 3$

算法 13 中描述了将多项式系数压缩到 10 比特以及从 10 比特解压缩的算法.代入$d = 10$求得被除数的取值上界为$(q \ll d) + \lfloor q/2 \rfloor = 7869184$,由此计算出$k$和$m$的最小取值分别为$k = 36$和$m = 8946684$.在 AVX2 实现中多次使用移位,当$k = 38$时可以减少一次移位,因此最终实现中使用了$k = 38$以及对应的$m = 35786735$.代入上述公式得到最终的实现公式,

$$x'' = \mathrm{Compress}_{7681}(x, 10) = \Big(\big((x \ll 10) + 3840\big) \cdot 35786735\Big) \gg 38$$

$$x' = \mathrm{Decompress}_{7681}(x'', 10) = (x'' \cdot 7681 + 512) \gg 10$$

和算法 12 不同的是,算法 13 中会出现 32 位乘法,AVX2 中的 32 位乘法指令只可以得到完整的 64 位乘积或者低 32 位乘积,不可以直接得到本算法实际需要的高 32 位乘积,因此算法中使用了一些 shuffle、blend 和 permute 指令用于调整数据在寄存器中的位置.

算法 13.10 比特压缩与解压缩的 AVX2 实现

输入:8 个多项式系数组成的a

输出:每个系数均压缩至 10 比特后的a

1: vpslld $a[i] \leftarrow a[i] \ll 10$

2: vpadddd$a[i] \leftarrow a[i] + 3840$

3: vpshufd$a1 \leftarrow a, 0xf5$

4: vpmuludq $a[i] \leftarrow a[i] \times 35786735$

5: vpmuludq $a1[i] \leftarrow a1[i] \times 35786735$

6: vpshufd $a \leftarrow a, 0xf5$

7: vpblendd $a \leftarrow a, a1, 0xaa$

8: vpshufb$a \leftarrow a, mask$

9: vpermq$a \leftarrow a, 0xf8$

输入:16 个系数均为 10 比特的a

输出:解压缩后的a

1: vpmul{l|h}w b[i],$a[i] \leftarrow a[i] \times q$

2: vpunpck{l|h}wdc, $b \leftarrow b, a$

3: vpadddc$[i] \leftarrow c[i] + 512$

4: vpaddd b$[i] \leftarrow b[i] + 512$

5: vpsrldc$[i] \leftarrow c[i] \gg 10$

6: vpsrld b$[i] \leftarrow b[i] \gg 10$

7: vpackusdw$a \leftarrow c, b$

值得一提的是,这里的多项式压缩与解压缩 AVX2 实现方法在重新代入参数调整后,可进一步用于优化 Kyber[13]、NewHope[10]等算法.

5 实验结果

本章给出 AKCN-MLWE 算法的 AVX2 实现性能数据及空间占用,本文使用原版 AKCN-MLWE 算法作为实验对比,即提交中国密码学会公钥密码竞赛第二轮的版本,包含了参考实现和 AVX2 实现.本文所设计实现的 AVX2 版 AKCN-MLWE 算法与原版所有算法参数(模数、维度、带宽等)保持一致.

本文进行测试的硬件环境为 2.30GHz(睿频加速最高 4.80GHz)八核 Intel Core i9-9880H 处理器和 16GB 内存,软件环境为 macOS 10.15 操作系统和 clang-11.0.3 编译器.为了保证性能数据的稳定性与准确性,测试前已关闭处理器的睿频加速(Turbo Boost)技术和硬件多线程(Hardware Multi-Threading)技术.编译参数主要使用了-O3、-march=native 和-mtune=native,用于开启编译器优化并且使用 AVX2.性能数据以 CPU 周期数为单位,可通过 CPU 主频换算成时间,计算公式为时间=周期数/频率.

AKCN-MLWE 算法的 KEM 性能对比见表 2.本文的 AVX2 实现中,密钥生成算法比参考实现提升 7.07 倍;密钥封装算法比参考实现提升 7.90 倍;密钥解封装算法比参考实现提升 11.78 倍;一次完整的 KEM(包含密钥生成、封装和解封装三个步骤)比参考实现提升 8.84 倍.表 2 中同时可以计算出本文

AVX2 实现的一次完整的 KEM 在基准测试环境下所需要的总时间约为,(53718+65150+53252)/2.30GHz=0.075ms.如果开启睿频加速技术,所需时间约为(53718+65150+53252)/4.80GHz=0.036ms≈0.04ms.

由表 2 可以看出本文提出的多种针对 AKCN-MLWE 的 AVX2 优化实现算法带来了巨大的性能提升,这主要归因于 AVX2 的强大并行性,使得我们可以将每个算法的多组数据存储到 AVX2 寄存器中,使得单条 AVX2 指令可以在单个时钟周期内同时处理多组数据的运算.本文对 AKCN-MLWE 中的多个核心算法,如多项式乘法,多项式加减法,约减算法,Con,与 Rec,压缩与解压缩,以及序列化与反序列化等,都采用了 AVX2 并行优化技术,这些优化技术是本文的 AVX2 实现相对于参考实现带来巨大性能提升的主要原因.

表 2AKCN-MLWE 算法不同实现版本的 KEM 性能比较

实现版本	密钥生成	封装	解封装
参考实现	433253	579736	680756
本文 AVX2	53718	65150	53252

[1] 单位均为 CPU 周期数

本文针对 AKCN-MLWE 算法的 AVX2 实现比众多 NIST 第二轮格密码算法的 AVX2 实现在相近安全级别下的性能更优,具体性能比较见表 3 与图 1.其中 Kyber[13]算法同样基于 MLWE 问题,其参数选取与 AKCN-MLWE 有一定的差异;NewHope[10]算法基于 RLWE,同样使用 NTT 与快速模约减进行加速;Saber[19]算法基于 MLWR,其多项式乘法使用 Toom-Cook/Karatsuba 进行加速.为了保证性能比较的公平性,Kyber[1]、NewHope[2]以及 Saber[3]的性能数据均使用其团队的 NIST 第二轮开源代码,并且在同一编译与测试环境下测量得出.

表 3 AKCN-MLWE 与 NIST 格密码的 AVX2 性能比较

算法名称	安全强度	密钥生成	封装	解封装
AKCN-MLWE	163	53718	65150	53252
Kyber768	181	53900	69862	57122
NewHope512	143	58388	90401	93418
Saber	203	88018	96336	90688

[1] 单位均为 CPU 周期数

[2] 安全强度均指经典安全强度

1 https://github.com/pq-crystals/kyber/tree/0b1b8945e472bcd822e5d55

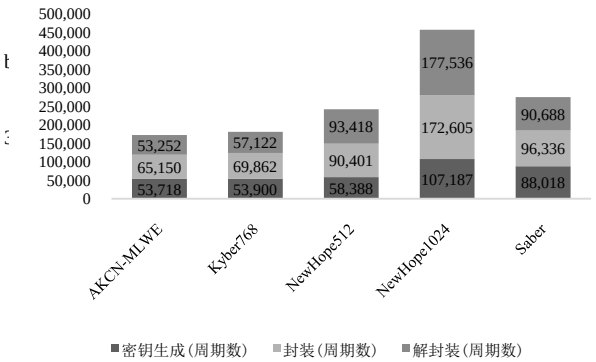

图 1 AKCN-MLWE 与 NIST 格密码的 AVX2 性能比较

通过表 3 和图 1 中的数据比较可得出,本文实现的 AKCN-MLWE 性能超过了相近经典安全强度下 NIST 算法 Kyber、NewHope、Saber.

6 结束语

本文给出了一种针对 AKCN-MLWE 算法的高效的 AVX2 实现方案,其中针对关键耗时的多项式乘法,采用删减一层的 NTT 并结合 Karatsuba 算法进行点乘,同时大幅减少了预计算表的空间占用;结合多种约减技术,充分降低了取模带来的性能开销;将算法中所有多项式运算并行化,进一步提升性能.实验结果表明,本实现方案较参考实现性能提升 8.84 倍,同时超过了众多 NIST 格密码算法.因此可以更好地满足高速的加解密需求,具有非常大的实际应用价值.

参考文献

[1] Shor P W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing, 1997, 26(5):1484-1509.

[2] Jin Z, Zhao Y. Generic and practical key establishment fromlattice//Proceedings of the17thInternational Conference on Applied Cryptography and Network Security. Bogota, Colombia, 2019: 302-322.

[3] Regev O. New lattice based cryptographic constructions//Proceedings of the 35th Annual ACM Symposium on Theory of Computing. San Diego,USA, 2003: 407-416.

[4] Regev O. On lattices, learning with errors, random linear codes, and cryptography//Proceedings of the 37th Annual ACM Symposium on Theory of Computing. Baltimore, USA, 2005: 84-93.

[5] LyubashevskyV, Peikert C, Regev O. On ideal lattices and learning with errors over rings//Proceedings of theAnnual International Conference on the Theory and Applications of Cryptographic Techniques.French Riviera, France, 2010: 1-23.

[6] Biasse J, Song F. Efficient quantum algorithms for computing class

groups and solving the principal ideal problem in arbitrary degree number fields//Proceedings of the Twenty Seventh Annual ACM SIAM Symposium on Discrete Algorithms. Arlington, USA,2016: 893-902.

[7]Cramer R, Ducas L, Peikert C, et al. Recovering short generators of principal ideals in cyclotomic rings//Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques. Vienna, Austria, 2016: 559-585.

[8] Cramer R, Ducas L, Wesolowski B. Short Stickelberger class relations and application to ideal-svp//Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques. Paris, France, 2017: 324-348.

[9]Langlois A, Stehlé D. Worst-case to average-case reductions for module lattices.Designs, Codes and Cryptography, 2015, 75(3):565-599.

[10]Alkim E, Ducas L, Pöppelmann T, et al. Post-quantum key exchange—a new hope//Proceedings of the25thUSENIXSecurity Symposium.Austin, USA, 2016: 327-343.

[11] Longa P, Naehrig M. Speeding up the number theoretic transform for faster ideal lattice-based cryptography//Proceedings of the 15th International Conference on Cryptology and Network Security. Milan, Italy, 2016: 124-139.

[12] SeilerG. Faster avx2 optimized ntt multiplication for ring-lwe lattice cryptography. IACR Cryptology ePrint Archive, 2018, 2018: 39.

[13]Bos J, Ducas L, Kiltz E, et al. Crystals-kyber: a cca-secure module-lattice-based kem//Proceedings of the2018 IEEE European Symposium on Security and Privacy. London, UK, 2018: 353-367.

[14]Lyubashevsky V, Seiler G. Nttru: truly fast ntru using ntt. IACR Transactions on Cryptographic Hardware and Embedded Systems,2019, 2019(3):180-201.

[15]Bertoni G, Daemen J, Peeters M, et al. Keccak//Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques.Athens, Greece,2013: 313-314.

[16]Fujisaki E, Okamoto T. Secure integration of asymmetric and symmetric encryption schemes//Proceedings of theAnnual International Cryptology Conference.Santa Barbara, USA, 1999: 537-554.

[17]Barrett P. Implementing the rivestshamir and adleman public key encryption algorithm on a standard digital signal processor//Proceedings of the Annual International Cryptology Conference.Santa Barbara, USA, 1986: 311-323.

[18]Montgomery P. Modular multiplication without trial division.Mathematics of computation, 1985, 44(170):519-521.

[19] D'Anvers J, Karmakar A, Roy S.Saber: module-lwr based key exchange, cpa-secure encryption and cca-secure kem//Proceedings of theInternational Conference on Cryptology in Africa.Marrakesh, Morocco, 2018: 282-305.

YANG Hao, Ph.D. candidate. His research interests include public-key cryptography and cryptographicengineering.

LIU Zhe, Ph.D., professor. His research interests include public-key cryptography and cryptographicengineering.

HUANG Jun-Hao, M.S. candidate. His research interests include public-key cryptography and cryptographicengineering.

SHEN Shi-Yu, M.S. candidate. Her research interests include post-quantum cryptography, cryptographicengineering and cryptographic protocols.

ZHAO Yun-Lei, Ph.D., professor. His research interests include post-quantum cryptography, cryptographic protocols and theory of computing.

Background

This paper focuses on the AVX2 implementation of the lattice-based AKCN-MLWE KEMscheme. This problem belongs to software optimization in cryptographic engineering. Lattice-based cryptography is the most promising category of post-quantum cryptography. The optimization workof lattice-based cryptography is very importantin replacing traditional public-key cryptography. Although some popular post-quantum cryptography schemes already have AVX2 implementations, there is still space for performance improvement. This paper presents a high-speed AVX2 implementation of AKCN-MLWE. The main contributions include:(1)summarize related state-of-the-art AVX2 optimization techniques from NIST post-quantum proposals, and apply them to AKCN-MLWE;(2)design a fast AVX2 linear polynomial multiplicationbased on Karatsuba; (3) design a highly-parallel polynomial compression and decompression algorithm. The results of this paper show that this work is8.84x faster than reference implementation, and also faster than

several NIST lattice-based schemes, such as Kyber and NewHope. The previous research directions of our research group include software optimization of elliptic-curve cryptography on 8-bit and 32-bit microcontrollersand GPU.