计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20210617
58(10): 2238-2252, 2021

Aigis 密钥封装算法多平台高效实现与优化

沈诗羽 何 峰 赵运磊
(复旦大学计算机科学技术学院 上海 200433)
(syshen19@fudan.edu.cn)

Multi-Platform Efficient Implementation and Optimization of Aigis-enc Algorithm

Shen Shiyu, He Feng, and Zhao Yunlei
(*School of Computer Science, Fudan University, Shanghai* 200433)

Abstract The new challenges brought by the rapid development of quantum computing technology have made post-quantum cryptography (PQC) a hot research topic in the current cryptographic community. The Aigis-enc key encapsulation mechanism is a post-quantum cryptographic algorithm based on the asymmetric module learning with errors (A-MLWE) problem, which is one of the algorithms that won the first prizes of public key cryptographic algorithms in the National Cryptographic Algorithm Design Competition held by the Chinese Association for Cryptologic Research. In order to resist quantum attacks, maintain the long-term security of national cyberspace, and contribute to the development of future national PQC algorithm standards, it is important to optimize the excellent post-quantum cryptographic algorithms developed by Chinese scholars. In this paper, we focus on optimizing the Aigis-enc algorithm for different platforms, including fast parallel implementation for high-performance platforms and compact implementation for embedded low-power platforms. Specifically, we fully optimize the existing AVX2 implementation of Aigis-enc using single instruction multiple data stream (SIMD) instructions, and provide its first lightweight compact implementation for the ARM Cortex-M4 platform. Our implementation includes the following optimizations: reducing the number of assembly instructions for Montgomery and Barrett reduction to improve the efficiency of reduction; using number theoretic transformations with trimmed layers and optimized instruction pipelining to speed up polynomial multiplication and reduce the precomputed table storage; providing a parallel implementation of assembly instructions for polynomial serialization and deserialization to speed up the processes of encoding, decoding and encryption; combining on-the-fly computation and space multiplexing to optimize the algorithm storage space. The experimental results show that the proposed optimization techniques can improve the original AVX2 implementation of the Aigis-enc-768 algorithm by 25% on an 8-core Intel Core i7 processor, and significantly reduce its precomputed table storage, code size and stack usage on the ARM Cortex-M4 platform, which is of great practical importance for future deployment of the algorithm.

收稿日期:2021-06-10;**修回日期**:2021-08-12
基金项目:国家自然科学基金项目(U1536205,61472084);国家重点研发计划项目(2017YFB0802000);上海市科技创新行动计划项目(16DZ1100200);上海市科学技术发展基金项目(16JC1400801);山东省重点研发计划项目(2017CXG0701,2018CXGC0701)
This work was supported by the National Natural Science Foundation of China (U1536205, 61472084), the National Key Research and Development Program of China (2017YFB0802000), Shanghai Science and Technology Innovation Development Program (16DZ1100200), Shanghai Science and Technology Development Funds (16JC1400801), and the Key Research and Development Program of Shandong Province(2017CXG0701, 2018CXGC0701).

Key words post-quantum cryptography; lattice cryptography; key encapsulation mechanism; AVX2 parallel optimization; embedded lightweight implementation

摘 要 量子计算技术快速发展带来的新挑战使得后量子密码(post-quantum cryptography, PQC)成为当前密码学界研究热点.基于格的密码方案因其安全高效的特性,已经成为后量子公钥密码的主流之一.Aigis 密钥封装算法(Aigis-enc)是我国学者自主设计的基于模格上非对称错误学习(A-MLWE)问题的后量子密码算法,是中国密码学会举办的全国密码算法设计竞赛公钥密码算法一等奖获奖算法之一.为了应对量子攻击,维护国家网络空间的长远安全,为未来国家后量子密码算法标准的制定和实际部署贡献力量,对我国自行研发的优秀后量子密码算法进行优化具有重要意义.工作重点关注 Aigis-enc 算法在不同平台的实现优化,包含高性能平台的快速并行实现与嵌入式低功耗平台的紧凑实现.具体而言,运用单指令多数据流(single instruction multiple data, SIMD)指令,充分优化了 Aigis-enc 现有 AVX2 实现,并提供了其首个 ARM Cortex-M4 平台的轻量级紧凑实现.实现包含 4 个关键优化点:降低 Montgomery 约减与 Barrett 约减汇编指令数目,提升了约减效率;使用裁剪层数的数论变换并优化指令流水调度,加速多项式乘法运算并减少了预计算表存储需求;提供了多项式序列化与反序列化的并行汇编指令实现,加快了编码解码与加解密过程;结合 on-the-fly 计算与空间复用优化算法存储空间.实验结果表明:提出的优化技术在 8 核 Intel Core i7 处理器上可将 Aigis-enc 算法原始 AVX2 实现提升 25%,且大幅减少了其在 ARM Cortex-M4 平台的预计算表存储、代码尺寸与运行堆栈占用,对算法的实际应用有重要现实意义.

关键词 后量子密码;格密码;密钥封装机制;AVX2 并行优化;嵌入式轻量级实现

中图法分类号 TP391

近年来,为应对大数据和互联网时代对计算能力的需求,各国都对量子计算研究投入了极大的力度和资源.谷歌、IBM、阿里、腾讯等公司相继成立了实验室来研究量子计算与后量子安全技术.谷歌于 2019 年研制出 53 位量子比特计算机[1],可于 3 min 20 s 内完成世界第一超级计算机 Summit 一万年才能完成的计算任务.2020 年 Yang 等人[2]研制出能在超过 1 K 温度下运作的量子计算平台,在实用性方面取得了巨大的突破.2021 年美国芝加哥大学和阿贡实验室研究人员利用超导同轴电缆提高量子态传输保真度,首次实现确定性多量子比特纠缠,为构建大规模量子计算机提供了一种模块化的方法[3].

量子计算主要通过利用量子叠加与纠缠两大量子物理学特性带来的强大并行处理性来超越经典计算的性能,且其计算模型中的特殊性质也给抗量子密码算法方案设计带来了巨大挑战.随着量子计算机的出现与量子计算技术的不断推进,现代密码算法面临着前所未有的挑战.传统的公钥密码设施大多基于离散对数或大整数分解困难问题,如联邦信息处理标准出版物(FIPS)186[4]中规定的数字签名方案和美国国家标准与技术局(NIST)特别出版物(SP)800-56A 和(SP)800-56B[5-6]中规定的密钥建立机制.这些公钥密码算法虽然目前尚无法被传统计算机攻破,但是均可在量子计算机中找到多项式时间的破解方法[7].密码技术是国家网络空间安全的关键技术,公钥密码体制在网络、金融、军事等方面都有着举足轻重的作用,它被广泛用于各种网络安全协议中,如 IPSEC,TLS 等.因此,实现抵抗量子攻击的新型密码体制,即后量子密码(post-quantum cryptography, PQC),是目前重要的研究方向.

为了应对量子计算的威胁,2016 年美国国家标准与技术局(NIST)开展后量子密码算法征集,向全球学者征集后量子密码算法[8].2019 年中国密码学会也举办了全国密码算法设计竞赛,旨在为我国制定后量子算法标准做准备[9].后量子密码的构造主要包括基于编码(Code-based)、基于哈希(Hash-based)、基于多变量(Multivariate-based)、基于格(Lattice-based)和基于同源(Isogeny-based)的.近期,NIST 宣布了第 3 轮 7 个进入决赛(finalists)的算法[10],其中 5 个是基于格构建的.相较于其他后量子密码技术路线,基于格的密码方案具有理论上的最坏情况困难保证[11],计算速度更快,更易于并行,而且通信开销较小,可用于构造多种功能强大的密码方案,如公钥加密、数字签名、全同态加密等.我国密码算法

设计竞赛一等奖 Aigis 方案[12-13]即是格上构建的基于非对称模-LWE(A-MLWE)与非对称 SIS(A-SIS)困难问题的方案,包含密钥封装机制 Aigis-enc 与数字签名方案 Aigis-sig. 对于密钥封装机制,相比于 NIST 第 3 轮中基于 MLWE 问题的 Kyber 方案,Aigis 密钥封装算法(简记为 Aigis-enc)需要基于额外的假设,但通过压缩公钥以及密钥和噪声从不同的分布中采样,Aigis-enc 在安全性与错误率之间取得了更好的权衡.

为了应对量子攻击,维护国家网络空间的长远安全,为未来国家后量子密码算法标准的制定和实际部署贡献力量,对我国自行研发的优秀后量子密码算法进行优化具有重要意义.本文重点关注 Aigis-enc 算法在不同平台的实现优化,包含高性能平台的快速并行实现与嵌入式低功耗平台的紧凑实现.对此,本文充分利用了单指令多数据(single instruction multiple data, SIMD)指令集,如 Intel 的 AVX2 指令集和 ARM Cortex-M4 的数字信号处理(DSP)指令集, 对算法底层多项式运算以及算法运行所需堆栈与存储空间进行优化. 特别地,据我们所知,本文工作提供了首个 Aigis-enc 的 ARM Cortex-M4 平台的轻量级紧凑实现. 具体而言,本文的主要贡献有 5 个方面:

1) 使用了改进版有符号数 Montgomery 约减与 Barrett 约减,并运用乘加指令减少了 ARM Cortex-M4 平台模约减所需指令条数,提升约减效率;

2) 使用了删减层数的数论变换(T-NTT)并提供 AVX2 与 ARM 的汇编实现,相比与传统 NTT,删减了变换的最后一层,极大地减少了预计算表存储需求;

3) 对多项式压缩并编码为字节流和字节流解码并解压缩这 2 个核心多项式序列化与反序列化操作,使用一个乘法和一个移位运算来替换耗时高的除法,并借助 AVX2 并行处理加快整个过程;

4) 调整算式结构以充分适应各平台指令特性,从而达到减少总指令数目与 load/store 指令开销的目的;

5) 采用 on-the-fly 计算与空间复用的优化方法,大幅减少了运行所需堆栈空间、代码大小以及存储占用.

实验结果表明:综合本文提及的优化技术,在 8 核 Intel Core i7 处理器上可将 Aigis-enc 算法原始 AVX2 实现提升 25%,且大幅减少了其在 ARM Cortex-M4 平台的预计算表存储、代码尺寸与运行堆栈占用,对算法实际应用有重要价值.

1 相关工作

基于格的密码体制首先由 Ajtai[14] 提出.2005 年 Regev[15] 提出的格上基于错误学习(LWE)问题的密码方案极大提升了格密码算法的效率,目前格密码已经成为密码学领域的研究热点.NIST 后量子标准化进程中,基于模格的密钥封装算法 Kyber 是第 3 轮入选算法之一.Zhang 等人[12] 观察到 MLWE 问题的私钥与噪音分布对方案安全性与错误率的影响不对等,从而提出了 A-MLWE 与 A-SIS 问题,并在此基础上构造了 Aigis 算法.其中,密钥封装机制 Aigis-enc 可视为 Kyber 的变体,但能更好地平衡安全性与错误率.自第 2 轮算法提交以来,Kyber 取消了对公钥的压缩,而 Aigis-enc 保留了该操作.

算法性能是衡量算法优劣的重要指标之一,近年来,学者们提出了很多针对格密码算法的软件优化实现方法[16],包含 AVX2 并行快速实现与 ARM Cortex-M4 平台的轻量级实现.2016 年 Alkim 等人[17]公开了基于 RLWE 困难问题的 NewHope 算法,算法参数设计使得其适合使用数论变换(number theoretic transform, NTT)加速多项式乘法,且使用浮点型 NTT 进行实现.Avanzi 等人[18-19] 于 2018 年公开了 Kyber 第 1 轮算法实现,包含 C 参考实现、C 优化实现与 AVX2 优化实现.该实现中使用了 $\mathrm{lb}\,q$ 层变换的整型 NTT.随后,在第 2 轮提交中,该团队对算法参数进行了调整,删减了 NTT 层数以降低 q 的规模[20].对于 ARM Cortex-M4 平台实现,pqm4[21] 提供了一个 Cortex-M4 微处理器上算法运行时钟周期与堆栈空间的测试框架.目前,Kyber 算法的 M4 实现已被纳入该项目.2020 年 Alkim 等人[22] 基于其 2016 年的工作[17],对 NewHope 算法的设计与 ARM 实现进行改进,并使用该技术优化了 Kyber 的堆栈使用.

目前尚未有针对 7681 模数的汇编指令实现.本文在充分结合运用这些优化技术的基础上,进一步调整代码运算结构,优化了多项式序列化、加解密过程,结合空间复用等方法,将指令个数与存储降至最低,从而给出了首个针对该模数高效紧凑的优化实现方案.

2 预备知识

本节给出了本文工作相关的预备知识.首先定义了变量符号、数学运算与格上困难问题,接着详细

描述 Aigis-enc 算法并介绍了实现平台与优化方式.

2.1 基本定义及符号说明

1) 环与多项式.令 n 是一个以 2 为底的正指数,q 是一个素数.定义集合 $Z_q=Z/qZ=\{0,1,\cdots,q-1\}$,$\mathcal{R}=Z[X]/(\Phi(X))$ 为分圆多项式环,其中 $\Phi(X)$ 是 n 维分圆多项式,$\mathcal{R}$ 中元素为整系数 n 维多项式.定义 $\mathcal{R}_q=\mathcal{R}/q\mathcal{R}=Z_q[X]/(\Phi(X))$,其中每个多项式系数均在环 Z_q 中.定义 $\mathcal{B}$ 为字符串空间.令 f 为 $\mathcal{R}_q$ 中的多项式,可表示为 $f=\sum_{i=0}^{n-1}f_iX^i$.多项式 f 的列向量形式是 $\boldsymbol{f}=(f_0,f_1,\cdots,f_{n-1})^{\mathrm{T}}$,行向量形式是 $\boldsymbol{f}=(f_0,f_1,\cdots,f_{n-1})$,其中 $f_i\in Z_q$,$i=0,1,\cdots,n-1$.

2) 运算符.默认情况下,常规字体字母表示 $\mathcal{R}$ 或 $\mathcal{R}_q$ 中的元素,黑斜体小写字母表示向量,黑斜体大写字母表示矩阵.对于一个元素 $x\in\mathbb{Q}$,$\lceil x\rfloor$ 表示 x 四舍五入至最近的整数.$|\cdot|$ 表示一个字符串的字节长度或者一个数的绝对值.mod q 表示模约减操作,$r'=r \bmod q$ 表示将整数 r 约减到 Z_q 中的代表元 r',$\bmod^{\pm}q$ 表示约减一个偶数至范围$(-\frac{q}{2},\frac{q}{2}]$(奇数至范围$[-\frac{q-1}{2},\frac{q-1}{2}]$).对于一个元素 $a\in Z_q$,$\|a\|_{\infty}^{\pm}$ 表示 $|a \bmod^{\pm}q|$.对于一个向量 $\boldsymbol{a}=(a_0,a_1,\cdots,a_{n-1})$,定义 $\|\boldsymbol{a}\|_{\infty}^{\pm}=\max_i\|a_i\|_{\infty}^{\pm}$.

3) 分布.对于集合 S,$s\leftarrow S$ 表示从 S 中随机选取一个元素 s.离散均匀分布是一种对称概率分布,其中有限数量的样本被采样的概率相同.中心二项分布 ψ_η 近似于高斯分布,该分布在区间$[-\frac{\eta}{2},\frac{\eta}{2}]$之间采样元素,$P[x|x\leftarrow\psi_\eta]=\frac{\eta!}{(\eta/2+x)!(\eta/2-x)!}2^{-\eta}$.

4) 算法参数.算法方案描述中,n 为多项式的维度,q 为模数,l 表示向量的维度,$\mathrm{lb}\,m$ 表示消息编码中 1 b 可编码结果的个数.此外,η 表示噪声的大小,d(或 t)表示 Z_q 域的整数被压缩(或删减)的比特数,$g=2^d$,其下标指示操作对象.pk,sk 与 ct 分别表示公钥、私钥与密文,B 为通信带宽,是公钥和密文长度的总和(单位为字节).方案分析中,δ 表示解密的错误率,$pq\text{-}sec$ 表示后量子安全等级,K 表示待封装密钥.

5) MLWE 困难问题. LWE 问题及其在模、环上的变体是格密码算法主要基于的困难问题.定义分布 $\chi_{x,\alpha}=\{(\boldsymbol{a},y=\boldsymbol{a}^{\mathrm{T}}x+z \bmod q):\boldsymbol{a}\leftarrow Z_q^n,x,z\leftarrow\psi_\alpha\}$,当秘密 x 从 Z_q 中均匀随机选取时,计算性 LWE 问题为给定多个服从 $\chi_{x,\alpha}$ 的样本,从中计算出 x.判定性 LWE 问题为给定多个 $\chi_{x,\alpha}$ 样本,区分其与 $Z_q^n\times Z_q$ 上的均匀分布.LWE 问题的困难性表现在没有多项式时间内的算法能解决这 2 个问题.标准格 LWE 困难问题允许灵活的参数选择,但在速度和存储空间上有明显的缺点,它涉及大量耗时的矩阵-向量乘法运算,且密钥和密文的存储需要较大的空间.基于 LWE 困难问题,RLWE 困难问题在效率和存储上有了很好的提升.主要的思想是通过构建一个额外的多项式环,把标准格中的 n 维向量替换成维度较小的多项式,具有稳定的结构特性,但额外代数结构的引入可能导致安全隐患.相比于 LWE 困难问题的 2 个形式,其不同便体现在样本为$(\boldsymbol{A},\boldsymbol{b}:=\boldsymbol{A}\boldsymbol{s}+\boldsymbol{e})$,且均匀样本随机选自 $\mathcal{R}_q^{l\times l}\times\mathcal{R}_q^l$,其中 $\boldsymbol{A}\leftarrow\mathcal{R}_q^{l\times l}$,$\boldsymbol{s},\boldsymbol{e}\leftarrow\psi_\eta$. MLWE 困难问题为 LWE 和 RLWE 问题的结合版本,平衡了安全与效率.MLWE 在多项式环结构上引入了向量维度 k,其值影响方案安全等级.特别的,当 $\boldsymbol{s}$ 和 $\boldsymbol{e}$ 是从不同的分布中得到时,可以提供 MLWE 问题的非对称版本(A-MLWE). A-MLWE 问题可看作是 MLWE 问题的一个特例,其中 $\boldsymbol{s}\leftarrow\psi_{\eta_e}$,$\boldsymbol{e}\leftarrow\psi_{\eta_e}$,$\eta_s\neq\eta_e$.

2.2 Aigis-enc 算法介绍

Aigis-enc 方案是基于 A-MLWE 困难问题的后量子密钥分装算法,其核心构造为 IND-CPA 安全的公钥加密 $PKE=(KeyGen,Enc,Dec)$,由密钥生成、加密和解密 3 个部分组成,遵循文献[23]的设计框架.在此框架基础上,通过 FO 转换(Fujisaki-Okamoto transformation)[24] 可构建 IND-CCA 安全的密钥封装机制 $KEM=(KeyGen,Encaps,Decaps)$.由于目前 FO 转换已有较为统一的模块化实现方案,且多为对 PKE 所涵盖函数接口的调用,所以本文重点关注 Aigis-enc 方案中 IND-CPA 安全的公钥加密模块,进而开展对底层多项式运算的优化实现.在文献[23]加密框架的基础上,私钥与噪音服从中心二项分布 χ_α,其值在区间$[-\alpha,\alpha]$内.Zhang 等人[12] 观察到,α 的值对方案安全性与正确性起着不对等的作用,换模压缩技术的使用也使得秘密向量及其对应的噪音向量在最终噪音项中起着不对等的作用.由此,Aigis-enc 中秘密向量与噪音向量分别从不同的分布 χ_s 与 χ_e 中采样,选取了非对称体制来平衡参数间的影响.

Aigis-enc-PKE 的密钥生成算法如算法 1 所示.密钥生成时首先从 n 位二进制串中均匀采样出 σ 和

ρ,然后将随机种子 ρ 传入 Sam 和 $Parse$ 函数以生成 NTT 域上的矩阵 $\boldsymbol{A}$,接着以 σ 为参数通过 CBD 算法生成私钥 $\boldsymbol{s}$ 和噪音向量 $\boldsymbol{e}$,最后计算 $\boldsymbol{As}+\boldsymbol{e}$ 后并压缩,与 ρ 拼接形成公钥 pk.

算法 1. Aigis-enc-PKE 密钥生成算法.

输出: 公钥 $pk:=(\boldsymbol{t},\rho)$,私钥 $sk:=\boldsymbol{s}$.

① function $Aigis\text{-}enc.KeyGen(\ )$

② $\sigma,\rho\leftarrow\{0,1\}^n$;

③ $\boldsymbol{A}\sim\mathbb{R}_q^{l\times l}:=Parse(Sam(\rho))$;

④ $(\boldsymbol{s},\boldsymbol{e})\sim\psi_{\eta_s}^I\times\psi_{\eta_e}^I:=CBD(\sigma)$;

⑤ $\boldsymbol{t}:=Compress_q(\boldsymbol{As}+\boldsymbol{e},d_t)$;

⑥ return $(pk:=(\boldsymbol{t},\rho),sk:=\boldsymbol{s})$.

⑦ end function

Aigis-enc-PKE 的加密算法如算法 2 所示.给定公钥 $pk=(\boldsymbol{t},\rho)$以及明文消息 $msg\in\{0,1\}^n$,使用相同的种子 ρ 生成矩阵 $\boldsymbol{A}$ 的转置.以随机数 r 为输入通过 CBD 生成错误向量$(\boldsymbol{r},\boldsymbol{e}_1,e_2)$,进而结合 $\left\lfloor\frac{q}{m}\times k\right\rceil$的明文内嵌入机制计算密文 $\boldsymbol{u}$ 和 v,最后将 $\boldsymbol{u}$ 压缩成 c_1,将 v 压缩成 c_2,得到密文 $ct=(c_1,c_2)$.

算法 2. Aigis-enc-PKE 加密算法.

输入:公钥 $pk:=(\boldsymbol{t},\rho)$、待加密消息 msg、随机数 r;

输出:密文 $ct:=(c_1,c_2)$.

① function $Aigis\text{-}enc.Enc(pk,msg,r)$

② $\hat{\boldsymbol{t}}:=Decompress_q(\boldsymbol{t},d_t)$;

③ $\boldsymbol{A}\sim\mathbb{R}_q^{l\times l}:=Parse(Sam(\rho))$;

④ $(\boldsymbol{r},\boldsymbol{e}_1,e_2)\sim\psi_{\eta_s}^I\times\psi_{\eta_e}^I\times\psi_{\eta_e}:=CBD(r)$;

⑤ $\boldsymbol{u}:=\boldsymbol{A}^{\mathrm{T}}\cdot\boldsymbol{r}+\boldsymbol{e}_1$;

⑥ $v:=\hat{\boldsymbol{t}}^{\mathrm{T}}\cdot\boldsymbol{r}+e_2$;

⑦ $v:=v+Decompress_q(msg,d_m)$;

⑧ $c_1:=Compress_q(\boldsymbol{u},d_u)$;

⑨ $c_2:=Compress_q(v,d_v)$;

⑩ return $ct:=(c_1,c_2)$.

⑪ end function

Aigis-enc-PKE 的解密算法如算法 3 所示.算法传入参数为密钥 sk 和密文 ct,通过对密文 ct,即 c_1 和 c_2 进行解压缩,分别得到 $\boldsymbol{u}$ 和 v,最后解码 $v-\boldsymbol{s}^{\mathrm{T}}\cdot\boldsymbol{u}$ 得到明文 msg.

算法 3. Aigis-enc-PKE 解密算法.

输入: 私钥 $sk:=\boldsymbol{s}$、密文 $ct:=(c_1,c_2)$;

输出: 解密消息 msg.

① function $Aigis\text{-}enc.Dec(\ )$

② $\boldsymbol{u}:=Decompress_q(c_1,d_u)$;

③ $v:=Decompress_q(c_2,d_v)$;

④ $msg:=Compress_q(v-\boldsymbol{s}^{\mathrm{T}}\cdot\boldsymbol{u},d_m)$;

⑤ return msg.

⑥ end function

2.3 实现平台与 SIMD 指令

SIMD 是一种采用一个控制器控制多个平行的处理单元、同时对一组向量化数据的每个元素执行相同操作的并行技术.相比于多进程并发运算,该技术实现的是空间上的数据级并行,同一时刻只有一个进程被执行.SIMD 技术多用于实现一些简单的通用运算,如算术运算、逻辑运算、数据排列混合、数据批量加载与存储等.

高级向量扩展(advanced vector extensions, AVX)指令集是 x86 架构微处理器中一种 128 b 的 SIMD 指令集,由英特尔提出,随后也得到了 AMD 的支持.AVX2 指令集是 AVX 的延伸,将大多数作用于整数上的指令扩展至 256 b,以增加一倍的运算效率.另外,三操作数运算指令的支持也减少了操作数的额外复制消耗.

ARM Cortex-M4 是 ARMv7E-M 架构的数字信号控制器,以满足高性能通用代码处理以及数字信号处理应用的需求.其核心通用指令集包含基本的 Thumb-1,Thumb-2 指令,以及 32 b 宽乘法.另外,ARM Cortex-M4 也提供了 SIMD 指令,即数字信号处理(digital signal processing, DSP)扩展指令集.该指令集可在 32 b 宽的寄存器内同时对 4 个 8 b 或 2 个 16 b 整数进行操作.

在格密码算法中,底层操作多为互相独立的多项式系数算术运算与逻辑运算,非常适合用 SIMD 指令进行并行优化.Aigis-enc 算法中多项式系数规模为 16 b,使用 AVX2 可实现 16 个数据的并行运算,DSP 可实现 2 个数据的并行运算,对多项式算数运算、模约减、数论变换等模块效率的提升有明显的效果.

3 Aigis-enc 算法测试及优化函数选定

本节给出了 gprof 与 nm 工具下 Aigis-enc-768 实现的软件分析测试结果,并据此选定开销大、调用频率高或存储占用多的模块以进一步优化,如离散采样、多项式乘法、模约减等;对于尚未并行优化的函数,如多项式序列化与反序列化,本文也将其纳入优化范围.对于选定的函数,本节中给出了相关理论与算法描述.

3.1 Aigis-enc 算法软件分析

Aigis-enc 算法团队在中国密码学会官网上提交了 Aigis-enc 原始实现,作为第 2 轮算法竞赛的支撑材料[12].为确定实现中有待进一步优化的函数、从而提升算法整体运行效率与空间占用,本文使用 gprof 与 nm 工具对 Aigis-enc-768 算法实现进行了测试分析,结果如图 1 所示:

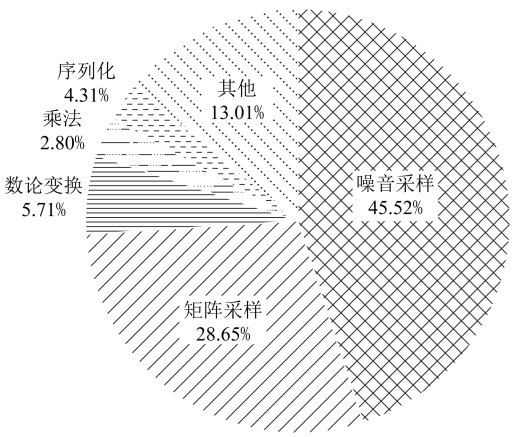

Fig. 1 Test results of the Aigis-enc-768 algorithm

图 1 Aigis-enc-768 算法测试分析

结果显示,伪随机数扩展占用了 61.70%的时间开销,主要用于噪音采样与矩阵 $\boldsymbol{A}$ 的生成,这 2 项运算所需时长分别为总体的 45.52%与 28.65%.一些多项式操作函数开销相比较大,如基于 NTT 的多项式乘法中,前向与逆向 NTT 共占 5.71%,多项式向量的点乘占 2.80%.此外,有 4.31%的时间用于多项式序列化与反序列化,包含多项式压缩与解压缩、消息编码与解码等多项式与字节流间的转换操作,其中部分函数目前尚未进行并行优化.对于 ARM 平台的轻量级实现,内存与栈空间占用为算法优化的重要指标之一.nm 工具得到的符号表显示,Aigis-enc-768 代码大小为 968.304 KB,其中,前向与逆向 NTT 分别占据了 30 KB.另外,为控制数据规模,模约减函数在运算中调用频率较高.原始实现中模约减并无显式调用,而是实现在各个函数中,这也会造成代码冗余.

本文对 Aigis-enc-768 算法优化主要集中于多项式运算处理模块,对通用哈希模块暂不进行调整.对于结构复杂的函数,如 NTT、模约减、压缩与序列化等,本文提供了汇编指令优化实现.结合工具分析结果,本文确定的主要优化点有 2 个方面:

1) AVX2.使用 NTT 变体提高多项式乘法计算效率;对未优化的多项式运算函数进行 AVX2 并行优化;提供部分函数的 AVX2 汇编指令实现,以优化流水调度;

2) ARM.减少 NTT 运算中预计算表大小;运用 on-the-fly[22] 等优化思想减少代码运行的栈空间;使用 DSP 指令减少运算所需时钟周期,并提升并行度.

3.2 模约减

多项式系数为群 Z_q 中的元素,将其规约为群 Z_q 中的代表元,可以控制系数规模,从而减小算法运行所需的时空资源.x86 与 ARM 架构的处理器通常使用除法指令来完成取模运算.为此,密码算法实现中引入了 Barrett 约减与 Montgomery 约减,以减少除法运算带来的过多消耗,同时保证约减可以在常数时间内完成,以抵抗侧信道攻击.

3.2.1 Barrett 约减

以 β 为基底,Barrett 约减可实现$[-\beta/2,\beta/2)$区间的整数 a 至群 Z_q 的规约,满足 $r=a \bmod q$,$0\leqslant r<q$.其约减过程可通过乘以模数逆元,将标准模约减所需的除法转化为乘法来完成,即:

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

为了更适配计算机硬件结构,Barrett 算法在实际部署中进行了一些调整[25-26],即将 $\lfloor a q^{-1}\rfloor$ 表示为 $\left\lfloor\frac{a}{\mu}\frac{\mu}{q}\right\rfloor$,通过预计算与移位运算来提高效率,其中 μ 一般表示为 2 的幂次.算法步骤见算法 4.

算法 4. 有符号数 Barrett 约减算法.

输入:16 b 有符号整数 a 满足 $-\beta/2\leqslant a<\beta/2$、模数 q 满足 $q<\beta/2$;

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

① function *Barrett*(a,q)

② $v=\left\lfloor\frac{2^{\lfloor\mathrm{lb}q\rfloor-1}\beta}{q}\right\rceil$;

③ $t=\left\lfloor\frac{av}{2^{\lfloor\mathrm{lb}q\rfloor-1}\beta}\right\rceil$;

④ $r=a-(tq \bmod \beta)$;

⑤ return r.

⑥ end function

算法 4 中,β 根据约减数据的规模来确定,对于 int16_t 类型的数据,通常设定 $\beta=2^{16}$.$v=\left\lfloor\frac{2^{\lfloor\mathrm{lb}q\rfloor-1}\beta}{q}\right\rceil$ 提供了一种 $\lfloor\mu/q\rceil$的近似,确定 q 后 v 为可预计算的常量.t 为约减 a 时需要减去的 q 的个数,通过 $a-tq$ 对其进行约减.

3.2.2 Montgomery 约减

不同于 Barrett 约减,Montgomery 约减作用于基数为 β 的 MONT 域上.数据需从正常域转换到 MONT 域才可使用 Montgomery 约减.

Montgomery 约减的主要思想是通过对 MONT 域内的数加上模数 q 的倍数,转换取模时除法的除数,使得整除运算更容易进行.约减完成后再转换回正常数域.对于无符号数,约减结果在$[0,2q)$范围内,需要与 q 进行比较判断,以得到$[0,q)$范围内的输出.有符号数的 Montgomery 约减在原本的算法上进行了一些调整,输入范围为$[-\beta q/2,\beta q/2)$,输出为$(-q,q)$范围内的有符号数,见算法 5.

算法 5. 有符号数 Montgomery 约减算法.

输入:32 b 有符号整数 a 满足$-\beta q/2\leqslant a<\beta q/2$;

输出:16 b 有符号整数 r'满足$-q<r'<q$.

① function *Montgomery*(a)

② $m=aq^{-1} \bmod^{\pm}\beta$;

③ $t=\left\lfloor m\dfrac{q}{\beta}\right\rfloor$;

④ $r'=\left\lfloor\dfrac{a}{\beta}\right\rfloor-t$;

⑤ return r'.

⑥ end function

3.3 多项式乘法与数论变换

数论变换是快速傅里叶变换(fast Fourier transform, FFT)在有限域上的一种特殊形式,常用于实现 Z_q 环上的多项式乘法加速.对于 n 长多项式,其中 n 为 2 的幂次,传统 NTT 要求参数 q 是满足 $2n|(q-1)$的素数.

令 ω 和 γ 分别为 Z_q 上的 n 阶与 $2n$ 阶单位根,$\omega=\gamma^2$,对于多项式 $f=\sum_{i=0}^{n-1}f_i x^i$,其前向 NTT 定义为

$$NTT:\mathcal{R}=Z_q[x](x^n+1)\to\mathcal{R}_q,$$

$$c\mapsto\tilde{c}=\sum_{i=0}^{n-1}\left(\sum_{j=0}^{n-1}c_j\gamma^j\omega^{ij}\right)x^i.$$

逆向 NTT 定义为

$$NTT^{-1}:\mathcal{R}_q\to\mathcal{R}=Z_q[x](x^n+1),$$

$$\tilde{c}\mapsto c=\sum_{i=0}^{n-1}\left(n^{-1}\gamma^{-i}\sum_{j=0}^{n-1}\tilde{c}_j\omega^{-ij}\right)x^i.$$

且二者满足 $f=NTT^{-1}(NTT(f))$.给定 f,$g\in Z_q[x]/(x^n+1)$,NTT 能够用来计算 $h=fg\in Z_q[x]/(x^n+1)$,矩阵形式的线性变换可表示为

$$\begin{pmatrix}\hat{f}_0\\ \hat{f}_1\\ \vdots\\ \hat{f}_{n-1}\end{pmatrix}=\begin{pmatrix}1&\omega&\omega^2&\cdots&\omega^{n-1}\\ 1&\omega^3&\omega^6&\cdots&\omega^{2n-2}\\ \vdots&\vdots&\vdots&&\vdots\\ 1&\omega^{2n-1}&\omega^{4n-2}&\cdots&\omega^{(2n-1)(n-1)}\end{pmatrix}\begin{pmatrix}f_0\\ f_1\\ \vdots\\ f_{n-1}\end{pmatrix}.$$

在 NIST 发起的后量子密码算法征集中,候选算法 Kyber 使用了一种 NTT 变体,删减了传统 NTT 最后一层[20],使得变换对单位根个数的需求降至原本的一半,从而参数只需要满足 $n|(q-1)$.对于这类遵循“底部裁剪”方法的数论变换,本文简称其为 $T\text{-}NTT$(truncated-NTT),并令 β 为删减的层数.Kyber 使用了 $T\text{-}NTT$ 的 $\beta=1$ 的情况.给定 $Z_q[x]/(x^n+1)$,对任意的整数 $\beta\geqslant 0$,q 是满足 $\dfrac{n}{2^{\beta-1}}|(q-1)$的素数,$T\text{-}NTT(f)$的一般形式为

$$T\text{-}NTT=\begin{pmatrix}\hat{f}_0+\hat{f}_1x+\cdots+\hat{f}_{2^\beta-1}x^{2^\beta-1}\\ \hat{f}_{2^\beta}+\hat{f}_{2^\beta+1}x+\cdots+\hat{f}_{2^{\beta+1}-1}x^{2^\beta-1}\\ \vdots\\ \hat{f}_{n-2^\beta}+\hat{f}_{n+1-2^\beta}x+\cdots+\hat{f}_{n-1}x^{2^\beta-1}\end{pmatrix}=$$

$$W_{\frac{n}{2^\beta}}\begin{pmatrix}f_0+f_1x+\cdots+f_{2^\beta-1}x^{2^\beta-1}\\ f_{2^\beta}+f_{2^\beta+1}x+\cdots+f_{2^{\beta+1}-1}x^{2^\beta-1}\\ \vdots\\ f_{n-2^\beta}+f_{n+1-2^\beta}x+\cdots+f_{n-1}x^{2^\beta-1}\end{pmatrix}.$$

通过 $T\text{-}NTT$ 变换,n 维多项式被分解为 $n/2$ 个一次子多项式.此时多项式向量之间的乘法为一次多项式的点乘.即对于变换后的 $\hat{f}=\sum_{i=0}^{n-1}\hat{f}_i x^i$ 与 $\hat{g}=\sum_{i=0}^{n-1}\hat{g}_i x^i$ 应执行 $n/2$ 对$(\hat{f}_{2i}+\hat{f}_{2i+1}x,\hat{g}_{2i}+\hat{g}_{2i+1}x)$的一次多项式点乘.

3.4 $\mathcal{R}_q$ 环上均匀采样

Aigis-enc 采用了确定性伪随机函数 Parse:$\mathcal{B}^*\to\mathcal{R}_q$ 扩展作为公钥一部分传输的种子 ρ,来生成通信双方计算所需的矩阵 $\boldsymbol{A}$. A-MLWE 困难问题要求该矩阵均匀随机,采用拒绝采样[27]可生成近似于均匀的分布.算法 6 描述了通过 Parse 函数采样多项式的过程.

算法 6. 多项式系数拒绝采样算法.

输入:字节流 $B=\{b_0,b_1,b_2,\cdots\}\in\mathcal{B}^*$;

输出:根据字节流采样的多项式 $\hat{a}$.

① function *Parse*(B)

② $i,j:=0$;

③ while $j<n$ do
④ $d:=b_i+256\times b_{i+1}$;
⑤ $d:=d \bmod^{+} 2^{\lceil \log_2 q \rceil}$;
⑥ if $d<q$ then
⑦ $\hat{a}_j:=d$;
⑧ $j:=j+1$;
⑨ end if
⑩ $i:=i+2$;
⑪ end while
⑫ return $\hat{a}_0+\hat{a}_1 X+\cdots+\hat{a}_{n-1}X^{n-1}$.
⑬ end function

3.5 多项式压缩与解压缩

基于 MLWE 困难问题构造的密钥封装方案,在实现中经常使用一些压缩和解压缩的方法,一方面,舍弃一些对正确性影响不大的低阶位可节省带宽和通信成本;另一方面,在加密和解密过程中执行 LWE 错误纠正.目前 Aigis-enc[11] 中使用的函数定义为

$$Compress_q(x,d)=\left\lfloor \frac{2^d}{q}x \right\rceil \bmod 2^d,$$

$$Decompress_q(y,d)=\left\lfloor \frac{q}{2^d}y \right\rceil \bmod q,$$

其中,$d<\lceil \mathrm{lb}q \rceil$为压缩后剩余的位数,$x\in Z_q$,$y\in Z_{2^d}$.$Compress$ 函数输入为 $x\in Z_q$,输出$\{0,1,\cdots,2^d-1\}$内一个整数,继而通过 $Decompress$ 可得到 $x'=Decompress_q(Compress_q(x,d),d)$,满足 $|x'-x \bmod^{\pm} q|\leqslant B_q:=\left\lceil \frac{q}{2^{d+1}} \right\rfloor$.

4 Aigis-enc 算法 AVX2 高效实现方案设计

本文提出的针对 Aigis-enc-768 算法的 AVX2 优化方案主要包括 3 个关键点:

1) 模约减.使用汇编指令实现针对有符号数的改进版 Barrett 约减与 Montgomery 约减,结合二者提升约减效率;

2) 多项式乘法.采取删减一层的 NTT,使用汇编指令实现并优化指令流水,以提升效率;

3) 多项式序列化处理.将压缩与解压缩中的除法运算替换为乘法与移位,从而可使用 AVX2 指令并行加速.

4.1 模约减

4.1.1 Barrett 约减 AVX2 实现

Aigis-enc-768 算法模数为 $q=7681$,从而可将系数规模控制在 16 b 有符号数范围内.设定 $\beta=2^{16}$,16 倍并行的 Barrett 约减算法可使用 4 条指令实现,具体见算法 7.

算法 7. 改进的 Barrett 约减算法的 AVX2 实现.

输入:向量化的 16 b 有符号整数 a、常数模数 q 以及预计算的 v 和 x;

输出:向量化的约减后的值 r.

① function *Barrett_AVX2*(a,q,v,x)
② vpmulhw $t\leftarrow a\,v$;
/* 计算 $a\,v$,取高 16 b */
③ vpsraw $t\leftarrow t\gg x$; /* 算术右移 x 位 */
④ vpmullw $t\leftarrow t\,q$;
/* 计算 $t\,q$,取低 16 b */
⑤ vpsubw $r\leftarrow a-t$;
/* 计算 $a-t$,得到结果 */
⑥ return r.
⑦ end function

算法的模数 q 固定后,输入参数 v 即为确定常量,可进行预计算并存储以节省计算量.计算 t 时,需要将 a 与 v 相乘的结果右移 $\lfloor \mathrm{lb}\,q \rfloor-1-\beta$ 位.结合 AVX2 指令特性,不需要存储完整的 32 b 结果,可以仅保留乘积高字并右移 $\lfloor \mathrm{lb}\,q \rfloor-1$ 位.同时,$tq \bmod \beta$ 的计算等价于 t 与 q 乘积低字,通过 vpmullw 指令可节省此处的取模运算.

4.1.2 Montgomery 约减 AVX2 实现

在 Aigis-enc-768 实现中,Montgomery 约减用于规约 Montgomery 域内 2 个 16 b 数的乘积,并保持其在 Montgomery 域内.根据模数设定 $\beta=2^{16}$,实现步骤见算法 8.

算法 8. 改进的 Montgomery 约减算法的 AVX2 实现.

输入:向量化的 32 b 有符号整数 a,记为 a_{hi} 和 a_{lo};

输出:向量化的 16 b 有符号整数 r'.

① function *Montgomery_AVX2*(a)
② vpmullw $m\leftarrow a_{\mathrm{lo}}\,q^{-1}$;
/* $m=(a_{\mathrm{lo}}\,q^{-1}) \bmod^{\pm}\beta$ */
③ vpmulhw $t\leftarrow m\,q$; /* $t=\lfloor m\,q/\beta \rfloor$ */
④ vpsubw $r'\leftarrow a_{\mathrm{hi}}-t$; /* $r'=a_{\mathrm{hi}}-t$ */
⑤ return r'.
⑥ end function

算法输入为 2 个 256 b 的 ymm 寄存器,分别用于存储向量化的 16 个 32 b 乘积的高字 a_{hi} 与低字 a_{lo},

其中,a 用于指代该乘积.同理,$m=a_{\text{lo}} \bmod^{\pm} \beta$ 的计算可通过仅保存低字来实现,$t=\lfloor mq/\beta \rfloor$ 可通过保留乘积的高字完成.最后,用 a_{hi} 减去 t 得到约减结果.

4.2 多项式数论变换改进

Aigis-enc 原始实现中采用了 AVX2 接口调用的形式,实现了 $(n,q)=(256,7681)$ 的传统 NTT.相比于 API 接口调用,直接使用 AVX2 指令实现具有更高的性能及指令调度安排灵活度,也可省略一些额外的调用消耗.另外,由于 Aigis-enc 三组参数中使用的 q 不统一,每个 q 对应的预计算表互不相同,导致算法需要开辟很多额外的存储空间.使用 *T-NTT*,根的个数即可由 n 降低至 $n/2$,减少了预计算表的空间需求.

根据分析,本文采用 *T-NTT*,删减 *NTT* 运算的最后一层,并提供了 AVX2 指令集形式的汇编实现.AVX2 实现中,先单独处理第 0 层,将系数 a_0,a_1,…,a_{63} 和 a_{128},a_{129},…,a_{191} 分别存入寄存器.通过指令 vpmullw 和 vpmulhw 进行蝴蝶变换,将系数与第 1 个单位根相乘,再通过指令 vpaddw 与 vpsubw 进行 Montgomery 约减,将系数约减至 $[-q,q]$ 范围内.多项式剩余的 128 个系数,即 a_{64},a_{65},…,a_{127} 和 a_{192},a_{193},…,a_{255},也进行相同的处理.不同于第 0 层,第 1~6 层中,256 个系数统一按序载入处理,乘以对应的单位根.在第 4~6 层中,需要使用 PACK 和 UNPACK 相关指令调整系数顺序,将相关的系数整合到一起.前向 *NTT* 采用了 CT 蝴蝶变换(Cooley-Tuckey butterflies),输入为正序多项式系数,输出为比特反转顺序的 *NTT* 域元素.逆向 *NTT* 采用了 GS 蝴蝶变换(Gentleman-Sande butterflies),输入为比特反转顺序的 *NTT* 域元素,输出正序多项式系数,其伪代码见算法 9 与算法 10.通过这 2 种变换的组合可以省略位反转操作,以提高运行效率.

算法 9. CT 蝴蝶变换 AVX2 实现.

输入:向量化的 16 个低位系数 rl、高位系数 rh、模数 q 与单位根 ζ;

输出:变换后的低位系数与高位系数向量 rh' 与 rl'.

① function *CT_Butterfly_AVX*2(rl,rh,q,ζ)

② vpmullw ζq^{-1},rh,t_1;
/* $t_1=(rh\times\zeta q^{-1})(\bmod 2^{16})$ */

③ vpmulhw ζ,rh,$(rh\zeta)_{\text{hi}}$;
/* $(rh\times\zeta)_{\text{hi}}=\left\lfloor \frac{rh\times\zeta}{2^{16}} \right\rfloor$ */

④ vpmulhw q,t_1,t_2; /* $t_2=\left\lfloor \frac{t_1 q}{2^{16}} \right\rfloor$ */

⑤ vpsubw t_2,$(rh\times\zeta)_{\text{hi}}$,$(rh\times\zeta)'$;
/* $(rh\times\zeta)'=(rh\times\zeta)_{\text{hi}}-t_2$ */

⑥ vpsubw $(rh\times\zeta)'$,rl,rh';
/* $rh'=rl-(rh\times\zeta)'$ */

⑦ vpaddw $(rh\times\zeta)'$,rh,rl';
/* $rl'=rh+(rh\times\zeta)'$ */

⑧ return (rh',rl').

⑨ end function

算法 10. GS 蝴蝶变换 AVX2 实现.

输入:向量化的 16 个低位系数 rl、高位系数 rh、模数 q 与单位根 ζ;

输出:变换后的低位系数与高位系数向量 rh' 与 rl'.

① function *GS_Butterfly_AVX*2(r)

② vpsubw rh,rl,rh'; /* $rh'=rl-rh$ */

③ vpaddw rh,rl,rl; /* $rl=rl+rh$ */

④ vpmullw ζq^{-1},rh',t_1;
/* $t_1=(rh'\times\zeta q^{-1}) \bmod 2^{16}$ */

⑤ vpmulhw ζ,rh',rh';
/* $(rh'\times\zeta)_{\text{hi}}=\left\lfloor \frac{rh'\times\zeta}{2^{16}} \right\rfloor$ */

⑥ vpmulhw q,t_1,t_2; /* $t_2=\left\lfloor \frac{t_1 q}{2^{16}} \right\rfloor$ */

⑦ vpsubw t_2,$(rh'\times\zeta)_{\text{hi}}$,$rh$;
/* $rh=(rh'\times\zeta)_{\text{hi}}-t_2$ */

⑧ return (rh',rl').

⑨ end function

4.3 多项式序列化与反序列化优化

本节主要说明了多项式压缩并编码为字节流、字节流解码为多项式并解压缩这 2 个过程的优化方案,在算法中用于对公钥与密文进行处理.优化的主要难点在于除法运算的处理,由于 AVX2 指令集中不包含除法指令,Aigis-enc 原始实现中尚未对其进行优化,这也造成了该过程较大的开销.

Barrett 在文献[25]中首次提出可以用一个乘法和一个移位运算来代替较为耗时除法运算,即

$$\left\lfloor \frac{a}{q} \right\rfloor = a\,b \gg s,$$

其中,$b=\left\lfloor \frac{2^s}{q} \right\rfloor$,$s>\text{lb}(aq)$.利用该转换思想,即可使用一条乘法指令和一条移位指令代替除法,从而向量化的数据可以借助 AVX2 指令实现充分的

16 倍并行加速.具体实施中,公钥 pk 与密文 c_1 需要压缩至 9 b,取值为 $b=35\,786\,735, k=38$;密文 c_2 需压缩至 4 b,取值为 $b=8\,737, k=26$.

5 Aigis-enc 算法 ARM 平台轻量级实现方案设计

本节介绍了针对 Aigis-enc-768 算法的 ARM Cortex-M4 平台优化方案.设计中采取与第 4 节相同的改进方案,并结合该平台寄存器大小和 Thumb 与 DSP 指令集结构,对实现方案进行调整,从而最优化计算的速度与开销.另外,轻量级实现平台还需考虑片上存储,本文采用空间复用等方法减少了堆栈与存储资源的占用.

5.1 模约减

5.1.1 Barrett 约减 ARM 实现

算法 11 描述了 Barrett 约减在 ARM Cortex-M4 上的实现,一次调用可完成 2 个 16 b 的约减.

算法 11. Barrett 约减算法 ARM 实现.

输入:封装 2 个系数的 32 b 寄存器单元 $a=a_{\mathrm{hi}}|a_{\mathrm{lo}}$;

输出:约减后的 32 b 寄存器单元 $r=r_{\mathrm{hi}}|r_{\mathrm{lo}}$.

① function *Barrett_ARM*(a)
② smulbb t_1, a, v; /* $t_1 \leftarrow a_{\mathrm{lo}} v$ */
③ smultb t_2, a, v; /* $t_2 \leftarrow a_{\mathrm{hi}} v$ */
④ asr $t_1, t_1, \#(\log(\beta)+\lfloor \mathrm{lb}\, q \rfloor -1)$;
/* $t_1 \leftarrow t_1 \gg (\log(\beta)+\lfloor \mathrm{lb}\, q \rfloor -1)$ */
⑤ asr $t_2, t_2, \#(\log(\beta)+\lfloor \mathrm{lb}\, q \rfloor -1)$;
/* $t_2 \leftarrow t_2 \gg (\log(\beta)+\lfloor \mathrm{lb}\, q \rfloor -1)$ */
⑥ smulbb t_1, t_1, q; /* $t_1 \leftarrow t_1 q$ */
⑦ smulbb t_2, t_2, q; /* $t_2 \leftarrow t_2 q$ */
⑧ pkhbt $t, t_1, t_2, lsl\ \#16$
⑨ usub 16r, a, t;
/* $r_{\mathrm{hi}} \leftarrow a_{\mathrm{hi}} - t_{\mathrm{hi}}, r_{\mathrm{lo}} \leftarrow a_{\mathrm{lo}} - t_{\mathrm{lo}}$ */
⑩ return r.
⑪ end function

算法的输入为封装在 32 b 单元的 2 个 16 b 有符号数 a_{lo} 与 a_{hi},输出为约减后的结果.v 预计算后作为常量写入寄存器,之后分别与 a_{lo} 和 a_{hi} 相乘、右移 $\mathrm{lb}\beta+\lfloor \mathrm{lb}\, q \rfloor -1$ 位后乘 q.将这 2 个 16 b 数封装后,运用 usub16 指令可同时执行 $a_{\mathrm{lo}}-t_{\mathrm{lo}}$ 与 $a_{\mathrm{hi}}-t_{\mathrm{hi}}$,并将结果存储回相应地址.

5.1.2 Montgomery 约减 ARM 实现

ARM Cortex-M4 平台上多数指令执行时间均为一个时钟周期.调整算式结构以结合乘加运算,则可充分利用此特性,将一个乘法与一个加法替换为乘加运算,从而节省了一个周期的时钟消耗.

本文应用了改进后的 Montgomery 约减[22].算法的输入为 2 个 16 b 的有符号数 a_{lo} 与 a_{hi},分别为 2 个 32 b 乘积的低字.输出为打包好的 2 个 $(-q, q)$ 区间内的约减结果.将预存储的 q^{-1} 调整为 $-q^{-1}$,则 $r'=a_{\mathrm{hi}}-t$ 变更为 $r'=a_{\mathrm{hi}}+t$,结合 t 计算中的乘法,即可使用 smlabb 指令在一个周期内完成 2 个 16b 的乘积与一个 32b 的加法.在 Aigis-enc-768 中,一个多项式向量由 3 个 256 维多项式构成,且 2 个多项式的乘法需要执行 1 856 次 Montgomery 约减(其中 2 次前向 NTT 需要 896 个,一次逆向 NTT 需要 448 个,向量点乘需要 512 个),改进后的方法可以大幅减少乘法所需的总时钟周期.

算法 12. Montgomery 约减算法 ARM 实现.

输入:封装 2 个系数的 32 b 寄存器单元 $a=a_{\mathrm{hi}}|a_{\mathrm{lo}}$;

输出:约减后的 32 b 寄存器单元 $r=r_{\mathrm{hi}}|r_{\mathrm{lo}}$.

① function *Montgomery_ARM*(a)
② smulbb t_1, a, v;
③ smulbb $r_1, t_1, -q^{-1}$;
/* $r_1 \leftarrow (t_1 \bmod \beta)(-q^{-1})$ */
④ smlabb r_1, r_1, q, t_1;
/* $r_{1\mathrm{hi}} \leftarrow \left\lfloor \frac{(r_1 \bmod \beta) q}{2^{16}} \right\rfloor + \left\lfloor \frac{t_1}{2^{16}} \right\rfloor$ */
⑤ smultb t_2, a, v;
⑥ smulbb $r_2, t_2, -q^{-1}$;
/* $r_2 \leftarrow (t_2 \bmod \beta)(-q^{-1})$ */
⑦ smlabb r_2, r_2, q, t_2;
/* $r_{2\mathrm{hi}} \leftarrow \left\lfloor \frac{(r_2 \bmod \beta) q}{2^{16}} \right\rfloor + \left\lfloor \frac{t_2}{2^{16}} \right\rfloor$ */
⑧ pkhbt r, r_2, r_1, asr #16;
/* $r \leftarrow (r_{2\mathrm{hi}} | (r_{1\mathrm{hi}} \gg 16))$ */
⑨ return r.
⑩ end function

5.2 多项式数论变换改进

本文使用了 $\beta=1$ 的 $T\text{-}NTT$,即删减最后一层.该方法大幅减小了预计算表存储空间,利于算法在嵌入式设备的部署.在 AVX2 实现中,由于寄存器存储空间充足,可加载完整的多项式并逐层变换.而

ARM Cortex-M4 为 32 b 架构,且资源受限(只有 16 个 32 b 寄存器),load 与 store 一次只能存取 2 个系数,若采取如 AVX2 实现般逐层变换的方法,会引入过多的 load 与 store 指令,所以需要对 *NTT* 结构进行调整以适应平台特性,调整后的方案如图 2 所示:

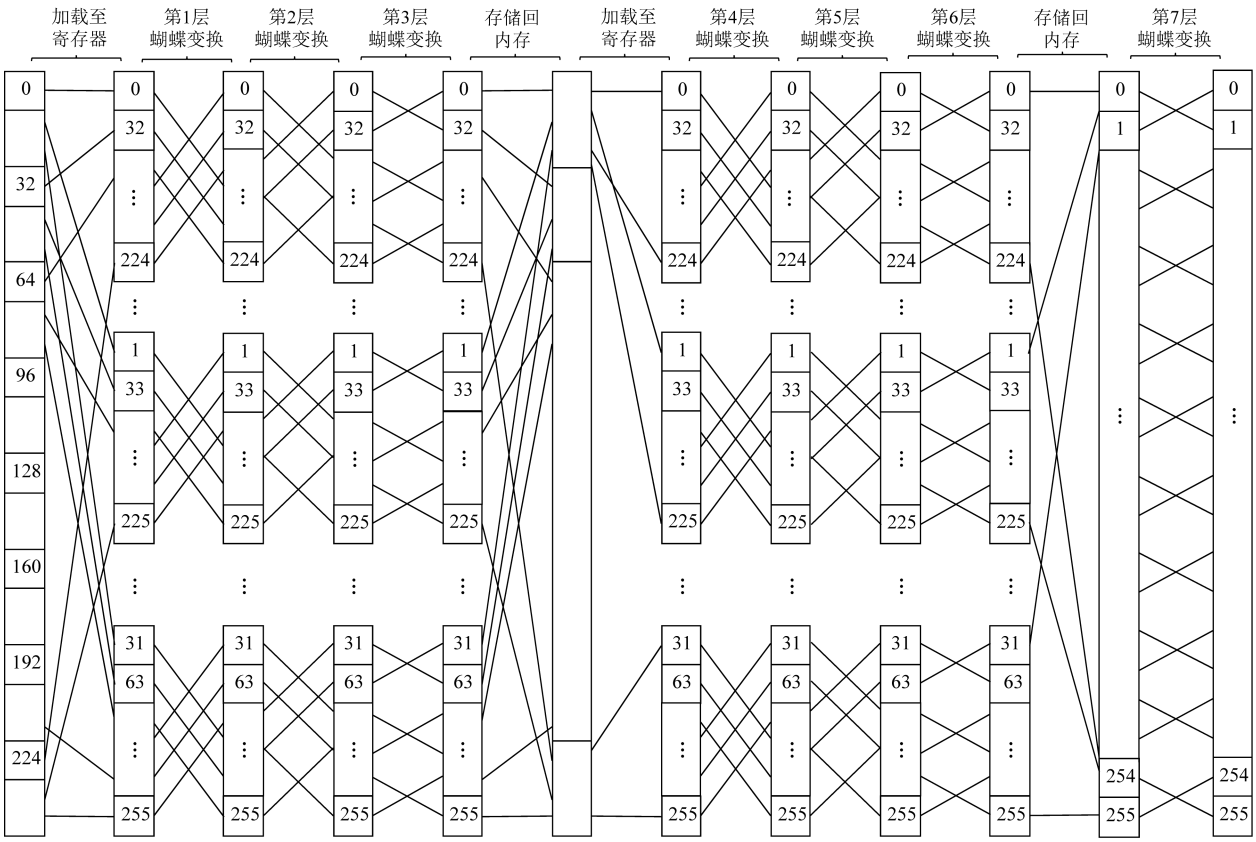

Fig. 2 Implementation of NTT on ARM Cortex-M4

图 2 ARM 平台 NTT 实现

该方案的主要思想为以固定的距离拆分多项式,每次取 16 个系数存入 8 个寄存器,进行 3 层变换后存回原位置.举例来说,第一次循环存取的系数为 $r_2=\{a_1 \parallel a_0\}$,$r_3=\{a_{33} \parallel a_{32}\}$,$r_4=\{a_{65} \parallel a_{64}\}$,…,$r_9=\{a_{65} \parallel a_{64}\}$.随后对其进行前 3 层数论变换,一对变换系数的距离间隔分别为 128,64,32.变换结束后存回原地址,地址指针前进 4 B,进入下一次循环.*T-NTT* 第 4~6 层的操作同理进行,一对变换系数的距离间隔分别为 16,8,4,最后统一执行系数间隔为 2 的第 7 层变换,得到最终结果.这里,一个寄存器可以存放 2 个 16 b 的系数,相当于二并行加速.

5.3 加解密并行优化处理

本节中论述了多项式压缩与序列化在 ARM Cortex-M4 上的优化方案.优化时考虑的要素主要为:

1) 调整算式结构以尽可能结合运算过程,减少所需指令的数目;

2) 尽量在一个系数存在于寄存器的整个生命周期内进行尽可能多的运算操作,以减少 load 与 store 指令的开销.

为了达成这 2 个目标,需要在效率与寄存器容量 2 个指标之间权衡折衷.加密中将秘密消息编码、多项式压缩与序列化结合,即为算法 2 中的行⑦⑨,$c_2:=Compress_q(v+Decompress_q(msg,d_m),d_v)$;在解密中结合多项式反序列化、解压缩与消息解码,即算法 3 中的行③④,从而得到解密后消息 $msg:=Compress_q(Decompress_q(c_2),\sigma)$.进而,本文结合除法运算转换法,对该运算顺序进行调整,使得 DSP 指令集中一些复合运算指令得以运用.优化后对单个位加密的核心过程见图 3,解密核心过程如图 4 所示.

这里加解密实现中均需对多项式进行预处理,通过指令 pkhbt 与 pkhtb 将 σ 与 k 同一索引位的系数封装入一个 32 b 的寄存器.继而,在加密中使用指令 smuad 与 mla,解密中使用指令 smlsd 与 mul

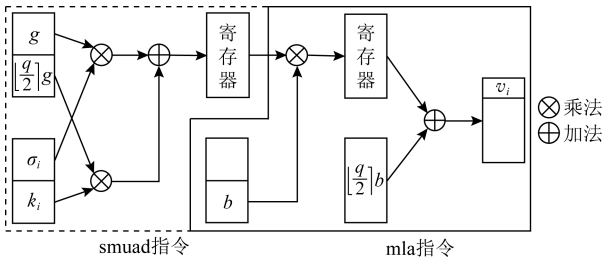

Fig. 3 Implementation of encryption on ARM Cortex-M4

图 3 ARM 平台加密核心步骤实现

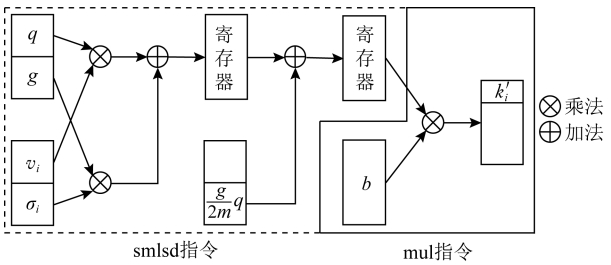

Fig. 4 Implementation of decryption on ARM Cortex-M4

图 4 ARM 平台解密核心步骤实现

以完成核心运算.此外,实现中还使用了位运算指令 bfi 与 orr,已实现寄存器中位的提取与比特流的拼接.

5.4 存储空间及栈空间优化

嵌入式设备上存储空间是一重要瓶颈,算法空间使用情况也决定了其能否实际部署.考虑到这一要素,本文在保持性能尽可能不受影响的情况下,采取 on-the-fly[22] 计算与空间复用等优化思想,减小代码大小以及运行所需栈空间.本文对 Aigis-enc-768 的优化中以堆栈使用和速度为主要指标,同时保持代码大小的合理性.

5.4.1 栈空间资源复用

在 Aigis-enc-768 原始实现中,对方案涉及的所有元素都申请了空间,并且分别对其进行采样、存储.考虑到该运算的线性特性,各对多项式之间的乘法以及多项式内各系数的乘法均互相独立,内存中不需要同时存在多个多项式.由此,本文在实现中减少了空间申请,通过空间复用的方式完成多个多项式运算.具体来说,对于密钥生成和密钥封装,申请一个多项式变量与一个多项式向量变量,对于密钥解封装,申请 2 个多项式变量,每个多项式的运算都在此空间上进行,得到结果后即存入对应的密钥或密文位置.

5.4.2 $\mathcal{R}_q$ 环上采样空间优化

基于模格的密钥封装方案在密钥生成与密钥封装中都包含 $\boldsymbol{A}\cdot\boldsymbol{s}$ 的核心运算,对此运算进行充分优化有利于降低代码运行空间需求.如 3.3 节所述,经过数论变换后 1 个多项式由 128 个一次多项式组成,它们之间的点乘互相独立,所以执行一次点乘仅需要一次多项式 $a_i+a_{i+1}x$ 与 $s_i+s_{i+1}x$ 相关的 4 个系数 a_i,a_{i+1},s_i 与 s_{i+1},这就意味着矩阵 $\boldsymbol{A}$ 的系数可以需要时再采样,而不用全部生成后再读取.详细地说,由种子 ρ 扩展得到字节流后,一次只采样 2 个符合条件的系数,与 $\boldsymbol{s}$ 中的 2 个系数点乘,并存储回 $\boldsymbol{s}$ 的相应位置,从而省略了存储矩阵 $\boldsymbol{A}$ 所用的空间.

与之相同的还有噪音的采样.原始实现中,噪音采样函数输入为字节流,输出为服从中心二项分布的多项式.采用 on-the-fly 计算优化思想[22],噪音采样函数输入调整为字节流和多项式,每次采样噪音后即与输入多项式对应位置的系数相加,而不用再进行存储.但是这种方法也意味着不能对噪音多项式进行数论变换(因为 NTT 作用对象为一个完整的多项式),从而 t' 的计算方式由 $\boldsymbol{t}'=\boldsymbol{A}\circ T\text{-}NTT(\boldsymbol{s})+T\text{-}NTT(\boldsymbol{e})$ 调整为 $\boldsymbol{t}'=T\text{-}NTT(T\text{-}NTT^{-1}(\boldsymbol{A}\circ T\text{-}NTT(\boldsymbol{s}))+\boldsymbol{e})$.这样会引入一个额外的 NTT^{-1} 的消耗.在 ARM Cortex-M4 平台上,这意味着需要增加 6944 时钟周期,约占总运行时间的 0.3%,但是可以节省矩阵 $\boldsymbol{A}$ 与多项式 $\boldsymbol{s},\boldsymbol{e}$ 共 7 680 B 的栈存储空间,权衡二者这样的消耗是值得的.

6 实验结果与性能比较

6.1 测试环境

本文针对 Aigis-enc 密钥分装机制设计了 AVX2 与 ARM Cortex-M4 的优化方案,并于 Aigis-enc-768 算法上实现验证.本节介绍了 2 个平台的代码测试环境.

1) AVX2 测试环境:本文进行 AVX2 测试的硬件环境为 3.6 GHz 的八核 Intel Core i7-9700K 和 32 GB 内存,且关闭了处理器的睿频加速技术(Turbo Boost)和硬件多线程技术(Hardware Multi-Threading);软件环境为 macOS Big Sur 11.1 操作系统与 Apple clang 12.0.0.31.1 编译器.使用的编译参数为-Wall -Wextra -Wpedantic -Wmissing-prototypes -Wredundantdecls -Wshadow -Wpointer-arith -mavx2 -mbmi2 -mpopcnt maes -march = native -mtune = native -O0 -fomit-frame-pointer -fno-stack-check.

2) ARM Cortex-M4 测试环境:本文使用带有 ARMv7E-M 指令集的 STM32F4DISCOVERY 开发板为测试平台,其内存为 196 KB,闪存为 1 MB,且最大频率为 168 MHz.

本文采用 CPU 周期数作为衡量算法效率的标准,AVX2 代码测试的 CPU 周期数为对应的函数执行 10 000 次后所得结果的中位数,ARM Cortex-M4 上的结果为执行 100 次后的中位数.

6.2 AVX2 实现性能比较

图 5 与图 6 为 Aigis-enc 与本文 AVX2 实现效率的对比,其具体数值分别如表 1 所示.

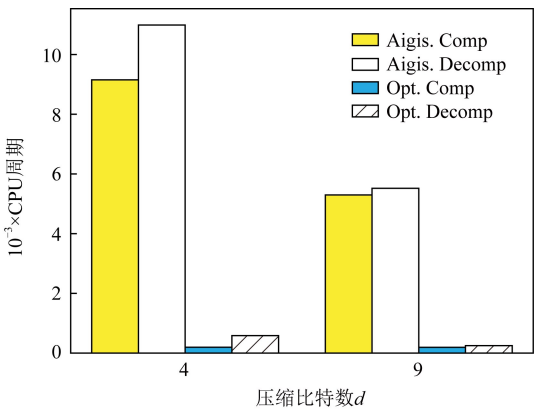

Fig. 5 Comparisons of polynomial compression and serialization betweenAigis-enc and our work

图 5 多项式压缩与序列化 AVX2 实现对比

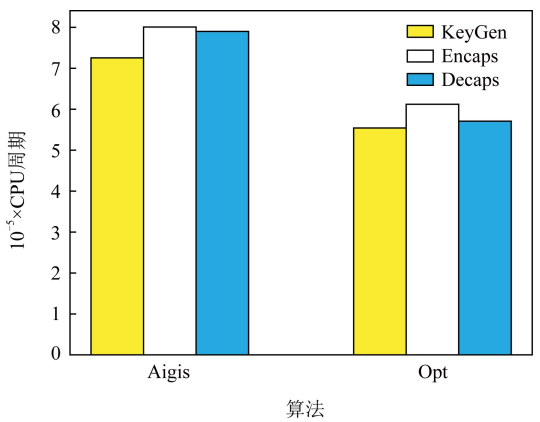

Fig. 6 Comparisons of Aigis-enc and our work in AVX2 implementations

图 6 Aigis-enc 与本文工作 AVX2 实现效率对比

图 5 为多项式压缩和序列化速度的比较,其中,压缩位数 $d=4$ 的情形用于密文 c 中第 2 部分 c_2 的生成,即计算 $c_2:=Compress_q(v+k,d)$,$d=9$ 用于生成密文的第一部分,即 $c_1:=Compress_q(\boldsymbol{u},d)$.本文设计中采取的方案性能将原始提升了 97.9% 和 94.7%,相当于对原过程有 47 倍和 19 倍的加速.

图 6 是 Aigis-enc 与本文工作 AVX2 实现效率对比.如表 1 中时钟周期数所示,本文工作将密钥生成函数提升 23.62%,密钥封装函数提升 23.56%,密钥解封装函数提升 27.74%,在总性能上体现为 25.00%的加速,彰显了本文优化设计方案的作用效果.

Table 1 Comparisons of Each Function of Aigis-enc and Our Work in AVX2 Implementations

表 1 Aigis-enc 与本文工作 AVX2 实现效率比较

测试函数		Aigis-enc /时钟周期	本文工作 /时钟周期	优化率/%
序列化	$d=4$	9 152	196	97.9
	$d=9$	10 988	584	94.7
反序列化	$d=4$	5 298	192	96.4
	$d=9$	5 520	250	95.5
密钥生成		725 454	554 105	23.62
密钥封装		800 740	612 110	23.56
密钥解封装		789 990	570 809	27.74
总性能		2 316 184	1 737 024	25.00

表 2 为 Aigis-enc 与本文 AVX2 实现 NTT 预计算表存储量的对比.在 $q=7\,681$ 下,为了适配 AVX2 指令实现,Aigis-enc 扩展的预计算表总共需要 3 008 B 存储空间,而本文工作仅需 1 584 B,减少了 47.34%的存储.

Table 2 Comparison of Precomputed Table Storage Between Aigis-enc and Our AVX2 Implementation

表 2 Aigis-enc 与本文 AVX2 实现 NTT 预计算表存储量对比

预计算变量	存储量/B		优化率/%
	Aigis-enc	本文工作	
ξ	1 504	792	
ξ^{-1}	1504	792	
总存储	3 008	1 584	47.34

6.3 ARM 实现性能与空间测试

鉴于目前 Aigis-enc 仅提供了 AVX2 实现,且尚未有 ARM 平台或 C 参考实现方案,这里仅给出本文优化前后 STM32F4DISCOVERY 开发板上的实验测试结果来对比,如图 7 所示,具体数值如表 3 所示.其中,优化前的算法为根据 Aigis-enc 原始实现自主修改的 C 实现.

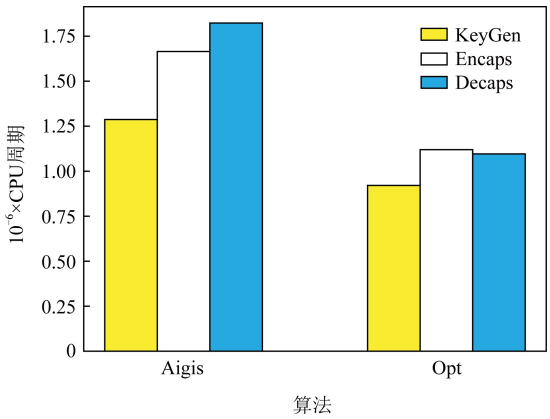

Fig. 7 Comparison of implementation before and after optimization of Aigis-enc on ARM platform

图 7 ARM 平台 Aigis-enc 优化前后实现效率对比

Table 3 Comparisons of Each Function of Aigis-enc and Our Work on ARM Cortex-M4 Platform

表 3 Aigis-enc 各函数与本文工作在 ARM Cortex-M4 平台上实现效率对比

测试函数	优化前/时钟周期	优化后/时钟周期	优化率/%
密钥生成	1 286 944	920 838	28.45
密钥封装	1 664 934	1 119 504	32.76
密钥解封装	1 822 945	1 095 917	39.88
总性能	4 774 823	3 136 259	34.32

在栈空间占用上,原始实现密钥生成、密钥封装及密钥解封装分别占据 10.8 KB,14.1 KB 与 15.3 KB,优化后达到 3.3 KB,2.9 KB 与 3.0 KB,总体上有 3.4 倍的提升,如表 4 所示:

Table 4 Menory Evaluation of Aigis-enc and Our Work on ARM Cortex-M4 Platform

表 4 Aigis-enc 与本文工作在 ARM Cortex-M4 平台上存储量测试

测试函数	存储占用/KB		优化率/%
	优化前	优化后	
密钥生成	10.8	3.3	69.44
密钥封装	14.1	2.9	79.43
密钥解封装	15.3	3.0	80.39
总性能	40.2	9.2	77.11

可见,采取改进的模约减算法、多项式乘法等优化方案,可大幅度提高 ARM Cortex-M4 上的实现效率.

7 总 结

本文给出了一种针对 Aigis-enc 算法的高效 AVX2 与 ARM Cortex-M4 实现方案,并选用 Aigis-enc-768 参数集进行实现验证.该实现方案在模约减、多项式乘法、序列化与反序列化等方面对 Aigis-enc 原始实现进行优化,同时大幅减少了代码尺寸、运行堆栈空间占用与预计算表存储需求,提升了算法总体性能,使其更易于实际部署,具有非常大的实际应用价值.

参 考 文 献

[1] Biance L Z. Google achieves "quantum hegemony"[J]. The Windy Generation, 2019, 2019(33): 40-41

[2] Yang C H, Leon R CC, Hwang J C C, et al. Operation of a silicon quantum processor unit cell above one kelvin [J]. Nature, 2020, 580(7803): 350-354

[3] Zhong Youpeng, Chang H S, Bienfait A, et al. Deterministic multi-qubit entanglement in a quantum network [J]. Nature, 2021, 590(7847): 571-575

[4] NIST. Digital Signature Standard (DSS)[S]. Gaithersburg: NIST, 2013

[5] NIST. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography [S]. Gaithersburg: NIST, 2018

[6] NIST. Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography [S]. Gaithersburg: NIST, 2019

[7] Shor P W. Algorithms for quantum computation: Discrete logarithms and factoring [C] //Proc of the 35th Annual Symp on Foundations of Computer Science. Piscataway, NJ: IEEE, 1994: 124-134

[8] NIST. Post-Quantum cryptography call for proposals [EB/OL]. (2017-01-03)[2021-05-01]. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals

[9] Chinese Association for Cryptologic Research. Public key algorithms selected to thesecond round competition of national cryptographic algorithm competitions [EB/OL]. (2019-09-27)[2021-05-01]. http://sfjs.cacrnet.org.cn/site/term/list_77_1.html (in Chinese)
(中国密码学会. 全国密码算法设计竞赛进入第 2 轮公钥算法[EB/OL]. (2019-09-27)[2021-05-01]. http://sfjs.cacrnet.org.cn/site/term/list_77_1.html)

[10] NIST. Post-Quantum cryptography round 3 submissions [EB/OL]. (2020-07-22)[2021-05-01]. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions

[11] Micciancio D, Regev O. Worst-case to average-case reductions based on Gaussian measures [J]. SIAM Journal on Computing, 2007, 37(1): 267-302

[12] Zhang Jiang, YuYu, Fan Shuqin, et al. Tweaking the asymmetry of asymmetric-key cryptography on lattices: KEMs and signatures of smaller sizes [J]. IACR Cryptol, ePrint Arch, 2019: No.510

[13] Zhang Jiang, YuYu, Fan Shuqin, et al. Supporting documentation of Aigis-enc [EB/OL]. (2019-09-27) [2021-05-01]. http://sfjs.cacrnet.org.cn/upload/5-db41c5175e9c.zip

[14] Ajtai M. Generating hard instances of lattice problems [C] // Proc of the 28th Annual ACM Symp on Theory of Computing. New York: ACM, 1996: 99-108

[15] Regev O. On Lattices, Learning with Errors, Random LinearCodes, and Cryptography [J]. Journal of the ACM, 2009, 56(6): 1-34

[16] Nejatollahi H, Dutt N, Ray S, et al. Post-Quantum Lattice-Based Cryptography Implementations: A Survey [J]. ACM computing surveys, 2019, 51(6): 1-129

[17] Alkim E, Ducas L, Pöppelmann T, et al. Post-quantum key exchange—A new hope [C] //Proc of the 25th USENIX Security Symp (USENIX Security 16). Berkeley, CA: USENIX Association, 2016: 327-343

[18] Avanzi R, Bos J, Ducas L, et al. Supporting documentation: CRYSTALS-Kyber: Algorithm specifications and supporting documentation [EB/OL]. (2017-12-21) [2021-05-01]. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/CRYSTALS_Ky-ber.zip

[19] Bos J, Ducas L, Kiltz E, et al. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM [C] //Proc of 2018 IEEE European Symp on Security and Privacy (EuroS&P). Piscataway, NJ: IEEE, 2018: 353-367

[20] Avanzi R, Bos J, Ducas L, et al. Supporting documentation: CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 2.0) [EB/OL]. (2019-01-30) [2021-05-01]. https://csrc.nist.gov/CSRC/media/Pro-jects/Post-Quantum-Cryptography/documents/round-2/submissions/CRYS-TALS-Kyber-Round2.zip

[21] Kannwischer M J, Rijneveld J, Schwabe P, et al. pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4 [R]. Gaithersburg: CSRC, 2019

[22] Alkim E, Bilgin Y A, Cenk M, et al. Cortex-M4 optimizations for {R, M} LWE schemes [J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020, 2020(3): 336-357

[23] Lindner R, Peikert C. Better key sizes (and attacks) for LWE-based encryption [C] //Proc of the Cryptographers' Track at the RSA Conf. Berlin: Springer, 2011: 319-339

[24] Fujisaki E, Okamoto T. Secure integration of asymmetric and symmetric encryption schemes [C] //Proc of the Annual International Cryptology Conf. Berlin: Springer, 1999: 537-554

[25] Barrett P. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor [C] //Proc of the Conf on the Theory and Application of Cryptographic Techniques. Berlin: Springer, 1986: 311-323

[26] Seiler G. Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography [J]. IACR Cryptol, ePrint Arch, 2018: No.39

[27] Gueron S, Schlieker F. Speeding up R-LWE post-quantum key exchange [C] //Proc of the Nordic Conf on Secure IT Systems. Berlin: Springer, 2016: 187-198

Shen Shiyu, born in 1997. PhD candidate. Her main research interests include lattice-based cryptography and cryptographic engineering.

沈诗羽,1997 年生.博士.主要研究方向为基于格的密码和密码工程.

He Feng, born in 1998. Master candidate. His main research interests include post-quantum cryptography and cryptographic engineering. (20210240069@fudan.edu.cn)

何 峰,1998 年生.硕士研究生.主要研究方向为后量子密码和密码工程.

Zhao Yunlei, born in 1974. PhD, distinguished professor at Fudan university. His main research interests include post-quantum cryptography, cryptographic protocols, theory of computing. (ylzhao@fudan.edu.cn)

赵运磊,1974 年生.博士,复旦大学特聘教授.主要研究方向为后量子密码、密码协议和计算理论.