一种面向全同态加密的低成本旋转因子发生器

付秋兴,李 伟*,别梦妮,陈 韬,杜怡然,南龙梅
(信息工程大学,河南郑州 450001)

摘 要: 全同态加密和后量子密码严重依赖数论变换(Number Theory Transformation,NTT)来加速多项式乘法,随着NTT多项式维度的提升,旋转因子的存储和传输对系统的影响愈发严重,为了解决这一问题,本文设计了一款面向全同态加密的低面积成本的旋转因子发生器. 本文首先分析了NTT算法的旋转因子调用规律,提出了一种动态旋转因子生成方案,通过数据生成和覆盖,将旋转因子压缩至原有存储空间的0.12%,在执行维度为65 536的NTT运算时,仅需78个单位的存储成本. 其次,本文基于全同态加密中RNS-CKKS方案,评估出了120个具有低汉明权重的60位宽的素数模,并设计了一种轻量级的Barrett模乘算法,基于该算法设计出了一种低面积成本模乘单元. 最后,本文基于动态旋转因子生成方案,实现了一款面向全同态加密的Radix-16 NTT蝶形单元的低成本动态旋转因子发生器,满足全同态加密中NTT运算的实际需求. 为进一步验证本文硬件设计的优越性,在Zynq UltraScale+XCZU9EG器件上进行实验验证,工作频率达252 MHz,Slice资源消耗降低了15%;在40 nm CMOS(ss-corner)工艺节点进行综合实现,与现有的设计相比,本文的硬件设计TPG(Throughput Per Gate)提升5倍以上.

关键词: 全同态加密;数论变换;旋转因子发生器;RNS-CKKS
基金项目: 国家自然科学基金(No.62302519)
中图分类号: TN402;TP309 **文献标识码:** A **文章编号:** 0372-2112(2025)08-2936-10
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250301

A Low-Cost Twiddle Tactor Generator for Fully Homomorphic Encryption

FU Qiu-xing, LI Wei*, BIE Meng-ni, CHEN Tao, DU Yi-ran, NAN Long-mei
(*University of Information Engineering*, *Zhengzhou*, *Henan* 450001, *China*)

Abstract: Fully homomorphic encryption and post-quantum cryptography rely heavily on NTT (Number Theory Transformation) to accelerate polynomial multiplication. As the dimension of NTT polynomials increases, the storage and transmission of rotation factors have an increasingly serious impact on the system. To solve this problem, this paper designs a twiddle factor generator with low area cost for fully homomorphic encryption. This paper first analyzes the calling rule of the twiddle factor of the NTT algorithm and proposes a dynamic twiddle factor generation scheme. Through data generation and overwriting, the twiddle factor is compressed to 0.12% of the original storage space. When performing the NTT operation with a dimension of 65 536, only 78 units of storage cost are required. Secondly, based on the RNS-CKKS scheme in fully homomorphic encryption, this paper evaluates 120 prime number modules with low Hamming weights and 60 bits wide, proposes a lightweight Barrett modular multiplication algorithm, and designs a modular multiplication unit with low area cost based on this algorithm. Finally, based on the dynamic twiddle factor generation scheme, this paper implements a low-cost dynamic twiddle factor generator for the Radix-16 NTT butterfly unit of fully homomorphic encryption, meeting the actual requirements of NTT operations in fully homomorphic encryption. To further verify the superiority of the hardware design in this paper, experimental verification was conducted on the Zynq UltraScale+ XCZU9EG device. The operating frequency reached 252 MHz, and the Slice resource consumption was reduced by 15%. The comprehensive implementation was carried out at the 40 nm CMOS (ss-corner) process node. Compared with the existing designs, the hardware design TPG (Throughput Per Gate) in this paper has increased by more than five times.

Key words: fully homomorphic encryption; number theoretic transformation; twiddle factor generator; RNS-CKKS
Foundation Item(s): National Natural Science Foundation of China (No.62302519)

收稿日期:2025-04-19;录用日期:2025-08-21;责任编辑:朱梅玉
*通讯作者:李伟

1 引言

随着人工智能技术的飞速发展,其在医疗、金融、交通、安防等领域的应用日益广泛. 在AI模型的训练和推理过程中,通常需要处理大量敏感数据,如个人健康信息、金融数据或商业机密. 传统的数据处理方法需要先解密数据,这可能导致隐私泄露的风险. 全同态加密(Fully Homomorphic Encryption,FHE)允许在数据保持加密状态的情况下进行计算,从而确保数据在整个处理过程中的隐私性. 2009年,Gentry构造出了第一个全同态加密算法[1],拉开了全同态加密算法的研究序幕. 虽然Gentry构造出了全同态加密所需的代数结构并可以满足同态计算的运算需求,但该算法在性能表现上十分糟糕,不具备实际的使用价值. 随着对全同态加密算法的研究不断深入,面向不同场景的具备实用性的算法被提出. 目前已有的主流全同态加密方案主要基于容错学习问题(Learning With Errors,LWE)以及其环上变体,包括BFV[2]、BGV[3]、CKKS[4]、GSW[5]、FHEW[6]和TFHE[7]等,这些方案的各自具有优点和局限性,尽管上述算法具备了一定的实用性,但FHE算法的密态运算仍然比明文运算慢4~5个数量级,这也是今天大规模部署FHE的关键障碍. 可行的方式之一是通过设计专用的硬件加速单元为算法执行提供大幅度加速,弥补大部分的性能差距.

全同态加密方案的构造严重依赖多项式运算,尤其是多项式乘法. 以CKKS算法为例,bootstrapping的实现需要在安全级别和多项式维度之间进行权衡,更高的安全级别需要维度更高的多项式来保证,现有的研究观察并建议多项式维度大于2^{16}的多项式才可以使得bootstrapping成为可能[8,9]. 在处理这种大型多项式时,多项式乘法的计算是昂贵的,数论变换(Number Theory Transformation,NTT)通过将多项式系数表示转换到点值表示,可以将多项式乘法的复杂度从$O(n^2)$降低到$O(n\log_2 n)$,因此,NTT算法在构造全同态加密方案,提高多项式运算的计算效率方面起着重要作用.

目前,NTT硬件方面的研究主要集中在对蝶形单元(Butterfly Unit,BU)和整体架构设计上,旋转因子的存储和调度问题经常被忽略. 事实上,在全同态加密中,多项式的维度通常在2^{16}以上,模一般在512 bit以上,旋转因子需要消耗2^{25} bit甚至更大的存储空间;受带宽限制,大量旋转因子的传输也会导致整体计算速度降低. 为了解决上述问题,本文设计了一款低成本的旋转因子发生器(Twiddle Factor Generation,TFG),其主要贡献如下:

(1)本文对消除了预处理的NTT算法中旋转因子之间的生成关系进行了分析,基于旋转因子之间的差值呈现出的递归对称性,设计了一种旋转因子动态生成方案,该方案以15个模乘单元和78个单位的存储器空间为代价,可以实现旋转因子的实时生成,能够以on-the-fly的形式支持维度在2^{16}及以下的Radix-16 NTT运算,节约了99.88%的存储空间.

(2)本文针对RNS-CKKS全同态加密方案,对NTT运算的模进行了评估,在保证安全性的条件下,选取了汉明权重为6的120个利于硬件实现的特殊模,并对Barrett模乘算法进行了分析和改进,基于此,实现了一款面向RNS-CKKS方案的低面积成本的Barrett模乘单元.

(3)本文基于上述旋转因子动态生成方案,针对全同态加密硬件设计中常用的Radix-16 NTT蝶形单元(Radix-16 NTT Butterfly Uint,RBU),增加了部分的存储空间简化控制逻辑,最终通过15个模乘单元和88个单位的存储空间实现了一款实用的Radix-16 NTT旋转因子发生器,能够实时地为RBU提供旋转因子.

2 算法介绍

多项式乘法可通过Cooley和Turkey提出的快速傅里叶变换(Fast Fourier Transform,FFT)将计算复杂度从$O(n^2)$降低到$O(n\log_2 n)$. 数论变换通过利用素数取模的周期性,在素数域上达到了和FFT相同的效果. 在全同态方案的有限域限制下,多项式乘法具体采用NTT方法来实现. 对于两个维度为n的多项式$\boldsymbol{a}=a_0+a_1x+\cdots+a_{n-1}x^{n-1}$和$\boldsymbol{b}=b_0+b_1x+\cdots+b_{n-1}x^{n-1}$,通过快速数论变换将多项式系数表示转化为多项式点值表示

$$\mathrm{NTT}(\boldsymbol{a})=A(k)=\sum_{i=0}^{n-1}a_i w^{ik} \bmod q \tag{1}$$

$$\mathrm{NTT}(\boldsymbol{b})=B(k)=\sum_{i=0}^{n-1}b_i w^{ik} \bmod q \tag{2}$$

在经过点乘操作后,最终通过快速数论变换的逆变换获得乘积多项式系数,即

$$\boldsymbol{c}=\mathrm{INTT}\big(\mathrm{NTT}(\boldsymbol{a})\odot\mathrm{NTT}(\boldsymbol{b})\big) \tag{3}$$

在分圆多项式环$Z_q/\Phi(x)$上实现NTT,当$\Phi(x)=x^n-1$时,要求旋转因子w使用q的$n-th$次单位根,即$w^n\equiv 1 \bmod q$,在执行NTT算法时,需要首先进行补零操作,这种NTT被称为正包裹卷积.

$$\begin{aligned}\boldsymbol{c}=&\mathrm{INTT}(\mathrm{NTT}(\mathrm{padding\_zero}(\boldsymbol{a}))\\&\odot\mathrm{NTT}(\mathrm{padding\_zero}(\boldsymbol{b})))\end{aligned} \tag{4}$$

目前大多数全同态密码方案中使用$\Phi(x)=x^n+1$,通过使用负包裹卷积(Negative Wrapped Convolution,NWC),利用$2n-th$次单位根$\psi^{2n}\equiv 1 \bmod q$,$\psi^n\equiv -1 \bmod q$,避免补零操作导致计算维度增加的问题. 在进行NTT时,需要首先计算$\tilde{\boldsymbol{a}}=\{a_0,a_1\psi,\cdots,a_{n-1}\psi^{n-1}\}$,而后对$\tilde{\boldsymbol{a}}$进行NTT运算. NWC引入了额外的预处理操作,增加了计算复杂度. 文献[10]通过改变旋转因子的调用次序,

直接使用$2n-th$次单位根ψ,将预处理操作融入NTT运算中,成功避免了N次乘法运算,进一步降低了运算复杂度,具体如算法1所示.

算法1 NTT算法

预计算:$w_k=\psi^{\mathrm{brv}(k)}$

输入:$a=\{a_0,a_1,\cdots,a_{n-1}\}$

输出:$a=\{a_0,a_1,\cdots,a_{n-1}\}$

1. $k=1$
2. FOR $l=n/2;l>0;l=l/2$ DO
3. FOR $s=0;s<n;s=j+1$ DO
4. FOR $j=s;j<s+l;j=j+1$ DO
5. $t=w_k\cdot a_{j+l}$
6. $a_{j+l}=a_j-t$
7. $a_j=a_j+t$
8. ENDFOR
9. $k=k+1$
10. ENDFOR
11. ENDFOR

在全同态加密算法中,多项式的维度n和模q不能独立选择,对于512比特的模q,多项式的维度一般会选取2^{16}及以上.由于硬件乘法器的功耗、面积与运算位宽的二次方成正比,直接实现大位宽的数论变换需要很大代价.因此,在硬件实现全同态加密时,会利用余数系统(Residue Number System,RNS),将大位宽的模q分解为一系列互素模$q=\prod_{i=0}^{L-1}q_i$,在分解后的各自的模域上分别进行NTT运算,因此在全同态加密进行NTT操作时,片内存储器需要存储L组2^{16}个旋转因子,消耗了大量的存储面积,同时也严重影响运算数据的传输.

3 面向Radix-16 NTT动态旋转因子生成方案

Radix-16 NTT的蝶形单元在全同态加密中被经常使用,为了使得所设计的旋转因子发生器能够更好地应用在全同态加密中,本节首先对算法1所使用的旋转因子顺序进行分析,而后针对Radix-16 NTT蝶形单元的实际需求来确定旋转因子的生成方案.

3.1 NTT旋转因子生成分析

基于算法1旋转因子的使用顺序,以N=16为例,图1列举了在执行NTT运算的4个stage中旋转因子使用情况,其中,框数字表示旋转因子的幂次,遵循由上到下,由左至右的使用顺序.在N=16的4个stage中,会分别使用到互不相同的1、2、4、8个旋转因子,且对于某个stage-i使用的第一个旋转因子的幂次为$2^{\log_2 N-i-1}$,其余旋转因子为每个stage的第一个旋转因子与之前所有stage使用的所有旋转因子的顺序乘积.

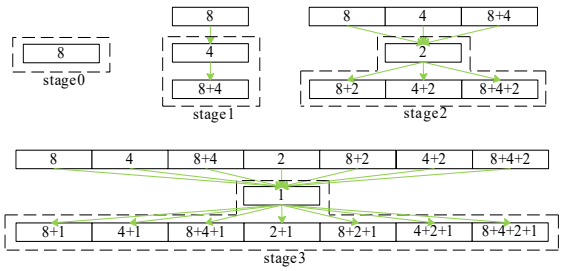

图1 NTT中旋转因子生成规律

在维度$N=2^n$的NTT算法中,依次使用到旋转因子$w[k]$,其中$k=0,1,\cdots,N-2$,共分为n个stage,设变量i表示stage的顺序,变量j表示每个stage中旋转因子的第j个旋转因子,可以推出k与i、j满足$k=2^i+j-1$,且$0\leqslant i<n$,$1\leqslant j\leqslant 2^i$,由此推出$k$与$i$、$j$具有唯一的对应关系$k=2^i+j-1$,$i=\lfloor\log_2 k\rfloor$,$j=k+1-2^{\lfloor\log_2 k\rfloor}$.

基于上述的观察,每个stage的第一个旋转因子幂次为2^{n-i-1},即$w[2^i]=2^{n-i-1}$,其余旋转因子为本stage的首个旋转因子与之前所有stage使用的所有旋转因子的顺序乘积,即$w[k]=w[2^i+j-1]=w[2^i]\times w[j-1]$,将$i$、$j$整体替换为$k$可得:

$$w[k]=\begin{cases}2^{n-\lfloor\log_2 k\rfloor-1}, & k=2^x\\ w\left[2^{\lfloor\log_2 k\rfloor}\right]\times w\left[k-2^{\lfloor\log_2 k\rfloor}\right], & k\neq 2^x\end{cases}\tag{5}$$

上述计算模型是基于图1观察直观得出,若依据该计算模型直接实现TFG,需要存储前$N/2$个旋转因子,来生成最后的stage的旋转因子,虽然存储规模降低了50%,但仍然消耗大量存储空间,与本文的设计目标相差甚远.主要原因是该计算模型实际上是以stage为一个生成节点,仅考虑到stage之间旋转因子的生成关系,并未充分利用stage内部旋转因子之间联系,未能使用数据覆盖操作以达到压缩存储空间的目的.上述计算模型虽然不是最优解,但提供了一些启发:若使TFG每次在至多使用一次模乘的条件下输出正确的旋转因子,则TFG需要提前存储至少$\log_2 N$个旋转因子,如幂次为1、2、4、8…的$\log_2 N$个旋转因子作为生成基,在此基础上,需要研究如何合理设计用于数据的暂存和覆盖的生成方案和对应的存储空间.

3.2 面向全同态加密的动态旋转因子生成方案

本节针对全同态加密所需的旋转因子生成方案进行分析和设计,根据NTT蝶形网络的运算层次,可将RBU划分为4个stage,每个stage分别需要旋转因子1、2、4、8个,考虑到全同态加密对硬件性能提出的较高要求,旋转因子的生成方案应当满足以下条件:

(1)该方案可以实现在BU运算时以on-the-fly的形式产生全同态加密所需全部15个旋转因子;

(2)旋转因子的生成逻辑需要支持硬件的流水操作来保证BU的性能,且各个旋转因子生成的路径长度需要保持一致,同一个周期内产生的旋转因子不具有前后生成的逻辑运算;

(3)旋转因子的生成逻辑控制简单,整体消耗的存储资源极小,TFG的设计不能为节约存储资源而浪费大量的其他硬件资源.

基于上述条件,本文旋转因子生成方案的设计思路是:提前预存部分旋转因子作为计算基底,通过存储器中的数据和模乘单元的输出共同作为模乘单元的输入,进行新的旋转因子生成. 由于该方案需要在每个周期能够实时产生15个旋转因子,因此需要使用到15个模乘(ModMult)单元,同时,考虑到全同态加密对硬件性能的较高需求以及上述各项条件,新的旋转因子需要在生成路径为一个模乘的条件下,完成相应的功能. 在RBU中的4个stage中,为了满足生成路径的和控制简单的条件,该方案的设计还应当遵循stage内的就地处理,即TFG产生的15个旋转因子w0~w14需要以w0、w1~w2、w3~w6、w7~w14为4组分别进行产生.

本节以N=256的NTT算法实现为例,针对TFG需要产生的15个旋转因子w0~w14中的w1~w2在stage-5中的生成逻辑进行分析,为了便于说明,本节所提到的存储器中的数字均指代以该数字为幂次的旋转因子,数字加法指代对应旋转因子之间的模乘操作. 在stage-5中w1~w2的生成过程中,在初始的4个周期(流水级数为4)需要从预先存储部分旋转因子的固定存储器中选取2个旋转因子输入到ModMult单元内,如图2所示,在提前存储幂次为1、2、4…128共8个旋转因子后,需要额外存储幂次为“128+64”、“128+32”、“128+64+32”来完成前4个周期的w1~w2共计8个旋转因子的产生. 4个周期后,当ModMult输出正确的旋转因子后,将输出数据返回并输入至ModMult单元,另外从固定存储器中选取1个旋转因子来共同计算所需的新的旋转因子. 基于上述分析,在提前存储幂次为1、2、4…128共8个旋转因子后,需要额外存储幂次为“128+64”、“128+32”、“128+64+32”来完成前4个周期的初始数据生成,4个周期后,将ModMult生成的旋转因子再次输入,并输入幂次为16的旋转因子(已标红),如图2所示,通过ModMult生成第5组w1~w2,第6组旋转因子由第2组数据模乘幂次为16的旋转因子来生成. 依照此逻辑,stage-5的数据生成实际需要8个不同幂次的旋转因子用于初始4个周期的数据生成,另外需要2个幂次为16和−8的旋转因子来完成后续生成,该方案支持4级流水的结构.

根据图2所示的旋转因子生成规律,在的NTT算法中,同一stage中相邻(或相隔)旋转因子的幂次差存在

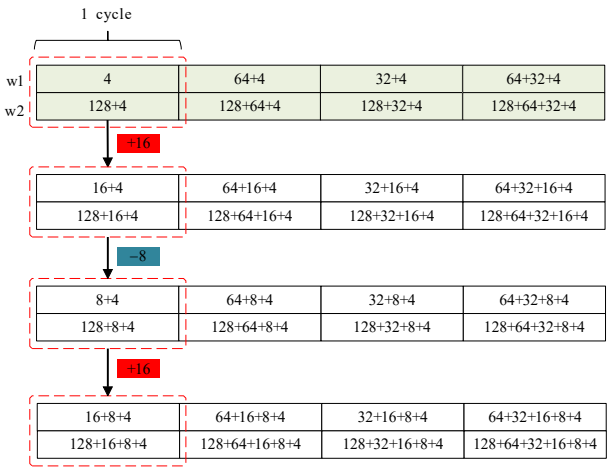

图2　NTT中旋转因子生成规律

递归对称性,本文的设计方案利用上述stage就地处理显示出的旋转因子幂次差的递归对称性,通过存储少量旋转因子,实现该stage的全部旋转因子生成. 为了进一步进行说明,本文将上述方案扩展至N=2^{16},并在图3中分别展示了NTT算法在stage-4、stage-8和stage-12所使用的16、256、4 096个旋转因子幂次差的具体情况,在stage的前4个周期的初始生成之后,旋转因子的幂次差以流水级数4为步长,整体呈现出递归对称性,对于周期计数cnt,该对称性与数据cnt/4的二进制表示的最右侧“1”的位置保持一致,具体地址生成逻辑如式(6)所示:

$$\mathrm{addr}=(\mathrm{cnt}/4)\&(-\mathrm{cnt}/4) \tag{6}$$

基于上述分析,w0的生成分为两个阶段,在初始生成阶段,每个周期需要从固定存储器中读取两个旋转因子输入到ModMult中;在后续的数据生成阶段,从存储器中读取上述存在递归对称性的某个幂次的旋转因子,与ModMult的输出寄存器的数据共同输入到ModMult单元. 图4展示了在w0的生成过程中,4个stage需要存储的所有旋转因子.

在图4中,每个stage左侧数据为stage初始生成阶段需要预存的旋转因子对应的幂次,右侧数据为stage数据生成阶段中所需的旋转因子对应的幂次. 蓝色标识为w0最终需要预先存储的旋转因子,黄色标识为w0在数据生成阶段需要预存的旋转因子,灰色标识为重复数据. 基于该方案,我们将生成逻辑进一步推广至w0~w14的产生,并对重复使用的旋转因子进行消除,最终在整个Radix-16 NTT的计算过程共需使用预存的旋转因子78个,其中,stage初始生成阶段所必需的预存的旋转因子42个,stage生成过程中所需的旋转因子40个(包含4个重复因子),旋转因子的压缩比例达99.88%.

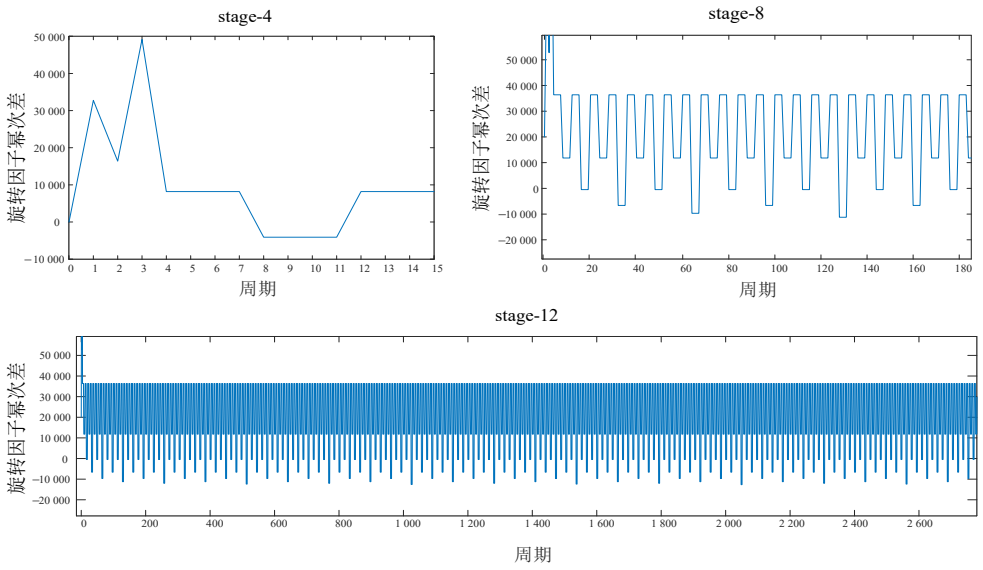

图3 stgae-4/8/12所使用的旋转因子之间幂次差

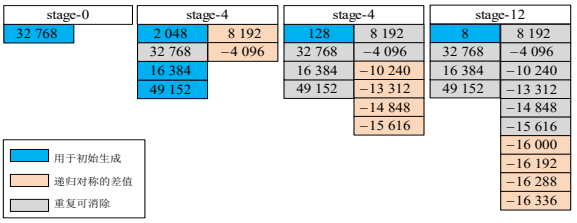

图4 N=65 536时w0产生所需的旋转因子

4 基于RNS-CKKS方案的低成本模乘单元

模乘是硬件单元中复杂且耗费资源的操作,在面向全同态加密的NTT单元实现过程中,需要使用到多个ModMult单元. 本节中本文使用FHE方案的特性来优化所提出的ModMult设计,使得本文的TFG设计不会为了节约存储空间而过多地消耗其他硬件资源,具体来讲,我们所提出的ModMult单元使用一类具有低汉明权重的特定模. RNS-CKKS方案[11]是一种基于余数系统的CKKS全同态加密方案,旨在提高近似计算的效率和性能,是一种专门用于处理浮点数和复数的同态加密算法,在人工智能应用的安全方面具有很大潜力. 目前,RNS-CKKS方案已经在多个开源库中得到了高效实现,例如HEAAN库和OpenFHE库. 实验结果表明,RNS-CKKS方案在处理近似计算时,相比传统的CKKS方案,具有更高的效率和更低的计算复杂度. 本节中介绍的优化技术主要针对RNS-CKKS方案,但也可应用于其他常规全同态加密方案.

在模乘单元的设计中,除法运算的实现是复杂的,Barrett模乘算法利用Barrett约减,通过预计算和移位操作来替代直接的除法运算,是一种高效的模乘算法. Barrett模乘利用浮点数的逼近和整数运算来替代直接的模运算,具体如算法2所示.

算法2 Barrett模乘算法

输入:A、B、q、$l=\lceil\log_2 q\rceil$、$T=\lfloor 2^{2l}/q\rfloor$

输出:$A\times B(\mathrm{mod}\ q)$

1. $U=A\times B$ //Full-Mult
2. $V=U\gg(l+1)$
3. $W=(V\times T)\gg(l+1)$ //Half-Mult
4. $X=W\times q(\mathrm{mod}\ 2^{l+1})$ //Half-Mult
5. $Y=U(\mathrm{mod}\ 2^{l+1})$
6. IF $(X<Y)$ $Z=X-Y+2^{l+1}$
7. ELSE $Z=X-Y$
8. IF $(Z\geqslant q)$ $Z=Z-q$
9. RETURN Z

传统的Barrett模乘算法中需要三个乘法运算,其中,包含1个Full-Mult和两个Half-Mult,Full-Mult输入两个n-bit,输出2n-bit,Half-Mult输入两个n-bit,输出n-bit. 在模乘单元的设计过程中,Full-Mult具有2个n-bit的随机输入,优化空间有限;所以本文的设计思路是:在RNS-CKKS方案中,找出特定性质的q和T,优化Barrett模乘算法中2个Halt-Mult,使用少量移位和加减运算来完成Halt-Mult操作.

CKKS算法本身是一种针对浮点数和复数近似计算的同态加密方案,通过重缩放(Rescaling)技术降低密文模的大小,从而提高计算效率. 在之前的某些工作中,ModMult单元的优化设计采用定制的约减方案,通过使用一个特定的模进行整体优化,而在128位的安全参数下,CKKS算法可以达到几十到几百层的乘法深度,因此在面向RNS-CKKS等FHE方案中,使用定制化的模乘设计显然是不合适的. 为了满足RNS-CKKS方

案的需求,我们从60位的数据中,筛选出足够数量的具备良好性质的模q,这些q应当满足以下条件.

(1)当多项式的维度为N时,素数模q具备$2N$个原根,来保证在模q条件下具备使用NTT进行加速的前提条件.

(2)q具备较低的汉明权重,为了优化RNS-CKKS方案中的ModMult单元,基于对算法3的分析,提出了对q的轻量级条件,即q的汉明权重是低的.

$$q=2^{l}+2^{sh_1}+2^{sh_2}+\cdots+2^{sh_{h-2}}+1 \tag{7}$$

$$HM_q=2+\sum sh_i \tag{8}$$

在此条件下,我们可以优化Barrett模乘算法中第二个Halt-Mult,使用少量移位和加减运算来完成Halt-Mult操作:

$$a\times q=(a\ll l)+(a\ll sh_1)+\cdots+a \tag{9}$$

此外,为了对Barrett模乘进行更全面的优化,要求素数q的逆缩放T同时具有低汉明权重符号二进制表达式,在ModMult设计中同样使用移位器和加法器取代第一个Half-Mult,以提升硬件整体性能,减少乘法资源的使用.

(3)符合条件的q的数量应当支持足够的乘法深度,这需要从低汉明权重进行穷举,不断放宽汉明权重限制筛选足够的模q以支持FHE运算. 在搜索出足够数量的q后,所有符合条件的q的最大汉明权重的最小值$\min\left(HM_{\max}\right)$,即为Half-Mult单元设计所需要支持的被加数个数,$\min\left(HM_{\max}\right)$越小,ModMult单元所消耗的硬件资源就越少. 从实际低汉明权重逐渐增加的数据遍历中,在每个固定汉明权重下,满足条件1、2的q的数量较少,导致了$\min\left(HM_{\max}\right)$偏大,无法达到预期的效果. 为了确保筛选出的$q$能够满足RNS-CKKS算法的位宽限制,需要保证$59\leqslant\lceil\log_2 q\rceil\leqslant 60$,为了降低$\min\left(HM_{\max}\right)$,我们将$q$表示为$q=2^{59}\pm2^{sh_1}\pm2^{sh_2}\pm\cdots\pm2^{sh_{h-2}}\pm1$,整体汉明权重小于等于6,该方法可以有效增加符合条件的模q的个数,且在硬件上基本不增加资源消耗. 在条件2的要求下,在Barrett算法中,在计算上要求$l=\lceil\log_2 q\rceil$,$T=\lfloor 2^{2l}/q\rfloor$,当$\lceil\log_2 q\rceil=59$,在$l>2\cdot sh_1+1$时,有不等式

$$\begin{aligned}&(2^{l}\pm2^{sh_1}\pm2^{sh_2}\pm\cdots\pm2^{sh_{h-2}}\pm1)\cdot(2^{l}\mp2^{sh_1}\mp2^{sh_2}\mp\cdots\mp2^{sh_{h-2}}\pm1)\\&=2^{2l}-(\pm2^{sh_1}\pm2^{sh_2}\pm\cdots\pm2^{sh_{h-2}}\pm1)^2>2^{2l}-q\end{aligned} \tag{10}$$

因此缩放倒数T可以表示为$T=2^{59}\mp2^{sh_1}\mp2^{sh_2}\mp\cdots\mp2^{sh_{h-2}}\mp1$,其中$l>2\cdot sh_1+1$. 利用这个条件,可以更有效地找到满足我们需求的模. 而当$q=2^{59}+2^{sh_1}\pm2^{sh_2}\pm\cdots\pm2^{sh_{h-2}}\pm1>2^{59}$时,$l=60$,$T$的形式不满足Barrett模乘算法中$T=\lfloor 2^{2l}/q\rfloor$,以汉明权重$HM_q=4$为例,通过数据遍历,符合条件的$q$仅有$2^{59}-2^{22}-2^{20}+1$、$2^{59}-2^{26}+2^{17}+1$、$2^{59}-2^{21}+2^{16}+1$共计3组满足条件(1)、(2)的数据,范围在$(2^{59}-2^{58},2^{59})$. Barrett模乘算法的核心思想是利用浮点数的逼近和整数运算来替代直接的模运算,对参数l和T的限制是为了确保算法的正确性和效率,在实际应用中,为了确保算法的正确性和效率,通常会遵守这些限制,但为了在60位宽中寻找出更多符合条件的模,本文在$q=2^{59}+2^{sh_1}\pm2^{sh_2}\pm\cdots\pm2^{sh_{h-2}}\pm1>2^{60}$,$l>2\cdot sh_1+1$的条件下,使用$l=59$而非$l=60$,因此需要对Barrett模乘算法进行分析和改进:

对于输入$U<q^2$,$q=2^{59}+2^{sh_1}\pm2^{sh_2}\pm\cdots\pm2^{sh_{h-2}}\pm1>2^{59}$,$q$的缩放倒数$T$满足$T=2^{59}-2^{sh_1}\mp2^{sh_2}\mp\cdots\mp2^{sh_{h-2}}\mp1$,其中$l=59$,且$l>2\cdot sh_1+1$. 在Barrett算法中,$U\bmod q=U-kq$,我们需要找到$k$的近似$t$.

令$T=2^{2l}/q-T_0$,其中$0<T_0<1$,则

$$t=\frac{UT}{2^{2l}} \tag{11}$$

令$q=2^{59}+2^{sh_1}\pm2^{sh_2}\pm\cdots\pm2^{sh_{h-2}}\pm1=2^{l}+\varphi$,则上述式为

$$\frac{U}{2^{2l}}\cdot T_0\cdot q=1+\frac{\varphi}{2^{l}}+\frac{\varphi^2}{2^{2l}}<2 \tag{12}$$

因此最终的计算误差范围在$2q$内,因此我们可以根据该误差对结果进行修正,改进为算法3.

算法3　改进型Barrett模乘算法

输入:A、B、q、$l=59$、$T=\lfloor 2^{2l}/q\rfloor$

输出:$A\times B(\bmod q)$

1. $U=A\times B$　　//Full-Mult
2. $V=U\gg(l+1)$
3. $W=(V\times T)\gg(l+1)$　　//Half-Mult
4. $X=W\times q(\bmod 2^{l+1})$　　//Half-Mult
5. $Y=U(\bmod 2^{l+1})$
6. IF $(X<Y)$　$Z=X-Y+2^{l+1}$
7. ELSE　$Z=X-Y$
8. IF $(Z\geqslant 2q)$ $Z=Z-2q$
9. ELSE IF $(Z\geqslant q)$　$Z=Z-q$
10. RETURN Z

基于上述形式,我们在60位宽的数据中进行筛选,由于以$q=2^{59}\pm2^{sh_1}\pm2^{sh_2}\pm\cdots\pm2^{sh_{h-2}}\pm1$的形式遍历数据时,高汉明权重可以任意表示低汉明权重数据,因此直接设置固定汉明权重,即可遍历出该汉明权重以下的所有数据. 在汉明权重为4、5、6时,搜索出符合条件的模q数量分别为5个、71个、236个. 为了使硬件能够更好地支持FHE应用,本文最终选取汉明权重为6的模q的集合. 为了更好地完成ModMult单元的硬件设计,本文从汉明权重为6的236个轻量级模中,对sh_1~sh_4的范围进行进一步限制来优化硬件中移位操作的范围,具

体是使每个sh_i的变换限制在+4范围以内，最终确定sh_1~sh_4最大值分别为28、27、23、21，最小值分别为25、23、19、17，相对位移4 bit，共有120个符合该条件的q，表1列出了其中部分q值.

表1 HM≤6的部分轻量级模q

编号	q_2	q_{10}
1	$2^{59}+2^{28}+2^{27}-2^{23}-2^{19}+1$	576 460 752 697 163 777
2	$2^{59}+2^{28}+2^{27}-2^{21}-2^{19}+1$	576 460 752 703 455 233
3	$2^{59}+2^{28}-2^{27}-2^{21}-2^{18}+1$	576 460 752 435 281 921
4	$2^{59}+2^{28}-2^{27}+2^{20}-2^{17}+1$	576 460 752 438 558 721
5	$2^{59}+2^{28}-2^{27}-2^{19}+2^{17}+1$	576 460 752 437 248 001
6	$2^{59}+2^{28}+2^{26}-2^{23}+2^{19}+1$	576 460 752 631 103 489
7	$2^{59}+2^{28}+2^{26}-2^{23}+2^{19}+1$	576 460 752 631 103 489
8	$2^{59}+2^{28}+2^{26}+2^{22}-2^{17}+1$	576 460 752 643 031 041

基于上述分析，我们使用$\min\left(HM_{\max}\right)=6$的120个模，并基于算法3，本文设计了如图5乘单元，通过移位和加减法替代了模乘算法中的两个Half-Mult单元，极大地节约了乘法资源，而不损害原RNS-CKKS方案的功能. 在安全性方面，通过使用Lattice-Estimator安全评估工具对本文提出的模进行了评估，根据CKKS方案，本文在维度$n=2^{12}$、标准差$\sigma=3.2$的条件下，对q_i进行了安全性评估，针对现有的各项攻击手段，其经典安全性

$$\text{class}=\beta\times 0.292 \tag{13}$$

以及量子安全性

$$\text{quantum}=\beta\times 0.265 \tag{14}$$

均能满足128 bit的安全性需求[12].

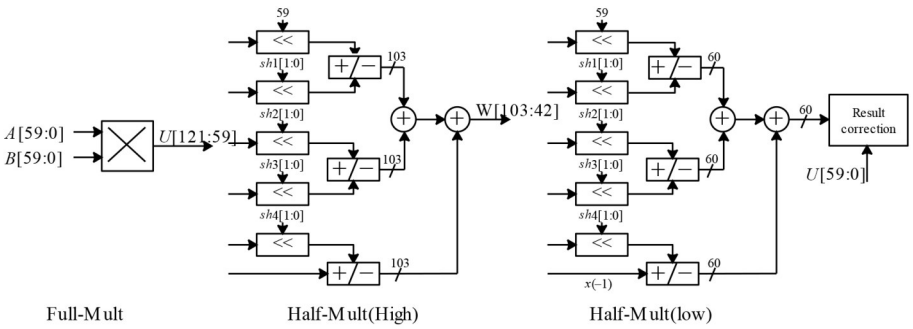

图5 改进后的Barrett模乘单元

5 面向全同态加密的动态旋转因子生成器

在第三节的旋转因子生成方案中，各个stage初始生成阶段所必须预存的旋转因子共计42个，stage生成过程中所需的旋转因子共40个(包含4个重复因子)，该方案的硬件实现需要存储78个旋转因子. 然而，为了使控制器的设计更加简单，本文增加少量的存储空间来存储部分的重复数据，并将所有需要预存的旋转因子进行划区，具体如图6所示. 该存储器内部划分6个存储bank，其中，bank0存储幂次为1、2…3、2 768的16个旋转因子，在旋转因子生成的过程中每次输出4个旋转因子；bank1存储了用于stage初始生成的31个旋转因子，为了方便控制每个stage的第一个旋转因子产生，bank1中额外增加了幂次为0的旋转因子(即数字“1”)，在每个stage的前4个周期中，每个周期输出15个旋转因子. bank2~bank5分别存储了w0、w1~w2、w3~w6、w7~w14生成阶段所需的10个旋转因子，根据stage的阶段以及计数器进行选择输出. Bank0~bank5实际消耗88个旋转因子的存储空间，虽然与之前的方案相比，该存储方案增加了10个单位的存储空间，但该方案在控制旋转因子生成的过程中，存储器的地址生成具有更简单的控制逻辑.

	Bank0		Bank1							
addr	addr_0	addr_1	addr_0	addr_1	addr_2	addr_3	Bank2	Bank3	Bank4	Bank5
0	1	256	0	4 096	2 048	6 144	8 192	4 096	2 048	1 024
1	2	512	32 768	36 864	34 816	38 912	−4 096	−2 048	−1 024	−512
2	4	1 024	16 384	20 480	18 432	22 528	−10 240	−5 120	−2 560	−1 280
3	8	2 048	49 152	36 864	51 200	55 296	−13 312	−6 656	−3 328	−1 664
4	16	4 096	8 192	12 288	10 240	14 336	−14 848	−7 424	−3 712	−1 856
5	32	8 192	40 960	45 056	43 008	47 104	−15 616	−7 808	−3 904	−1 952
6	64	16 384	24 576	28 672	26 624	30 720	−16 000	−8 000	−4 000	−2 000
7	128	32 768	57 344	61 440	59 392	63 488	−16 192	−8 096	−4 048	−2 024
8							−16 288	−8 144	−4 072	−2 036
9							−16 336	−8 168	−4 084	−2 042

图6 分Bank旋转因子预存储情况

本文设计的Radix-16 NTT动态旋转因子生成方案支持4级的流水线设计,即通过模乘生成新的旋转因子时,ModMult乘单元硬件结构的最大流水级数是4,在第4节中的低成本的ModMult乘单元的设计中,本文通过模的筛选优化了模ModMult单元,通过使用加/减和移位操作对两个Half-Mult进行了替代,但在结果修正中增加了判断和减法操作. 为了提高硬件整体性能,还需要对模乘单元进行合理的流水划分. 在ModMult单元中,60比特乘法操作部分的关键路径最长,因此,本文将该操作拆分为4个独立的30比特乘法以及2层加法操作,并与后续两个Half-Mult操作和结果修正部分进行了如图7所示的流水级划分.

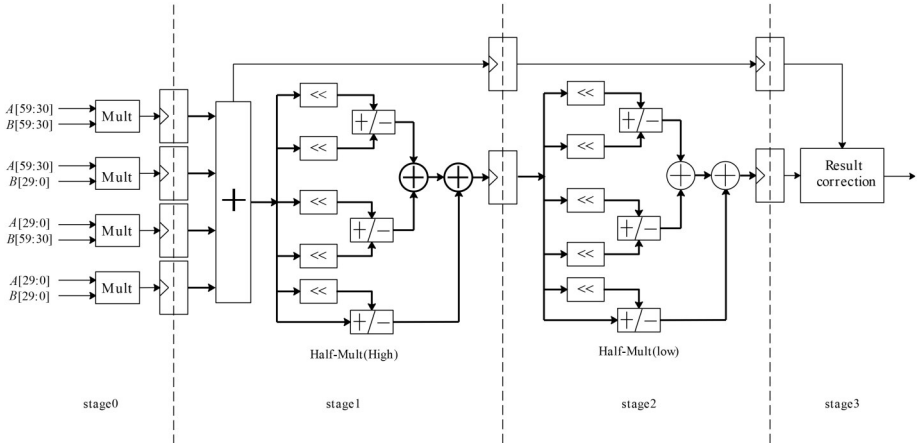

图7　低成本模乘单元的流水级划分

基于对旋转因子的预存储和低成本模乘单元的设计,本文设计了如图8所示的面向全同态加密的NTT整体运算结构,该结构由旋转因子发生器、Radix-16 NTT蝶形运算网络和控制组成. 其中,旋转因子发生器分为存储和计算两部分,存储部分包含图6所示的6个存储bank,计算部分包含15个ModMult单元,分别对应w0~w14的旋转因子产生,且由控制器具体控制旋转因子发生器内多Bank的存取和模乘单元的输入选择;RBU包含32个蝶形单元,每个单元由1个ModMult、1个模加和1个模减单元组成,同样构成4级流水结构,该网络整体构成16级流水.

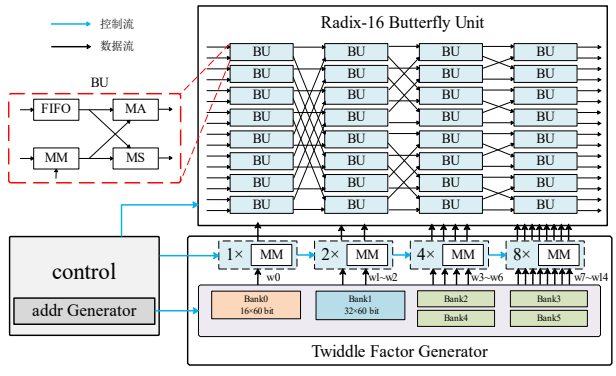

图8　Radix-16 NTT运算单元

在面向多项式维度为2^{16}的NTT计算时,由控制器提前4个周期控制TFG实时产生15个所需的旋转因子并分发到对应的蝶形单元内,在计算的过程中,NTT算法中的每个stage消耗时钟周期2^{12},整体消耗65 556个时钟周期、15个ModMult单元以及88个旋转因子的存储空间.

6　实验结果分析

本文利用Verilog HDL分别在Xilinx Vivado(v2019.1)和综合工具上进行综合实验并与近些年相关的研究进行了对比和分析. 本文首先通过Vivado2019.1软件进行仿真综合,选取芯片型号为Virtix-7中的Zynq UltraScale+ XCZU9EG,最高工作频率达252 MHz,表2具体展示了本文的TFG设计在FPGA的实验中,与不同文献中旋转因子发生器的设计的对比情况. 在ASIC的对比实验中,现有研究往往给出NTT的整体架构的实验结果,因此,为了体现本文设计的优势,本文基于第四节中低成本模乘单元的设计实现了图8所示的NTT的整体架构,并基于40 nm CMOS(ss-corner)工艺库完成了布局布线,表3具体列出了本文的设计在ASIC后仿的性能参数与近几年相关研究成果的比较.

在针对TFG的设计实验当中,文献[13]对针对维度N为$2^{16}/2^{17}$的NTT运算设计了旋转因子发生器,使用到了FIFO进行周期延迟,数据压缩比例为3.13%,与该文献相比,本文的设计在TPS方面提升约20%,旋转因子的存储仅消耗原空间的0.13%,在存储空间方面降低了一个数量级,达到了节约存储空间的设计目的. 文献[14,15]分别提出NTT旋转因子发生器的FPGA实现方法,其中,文献[14]设计的并行on-the-fly TFG针对串行NTT的不同的stage分别提供旋转因子,通过feedback进

表2 FPGA实验结果对比

对比文献	器件	n	Radix	$\log_2(q)$	Freq	LUT/FF/DSP/BRAM	Slice	Throu(Gbps)	TPS(Mbps/Slice)
文献[13]	U250	$2^{17}/2^{16}$	16	60	250	24 398/17 475/150/25	13.6 k	59.40	4.37
文献[14]	KCU105	2^{15}	—	64	150	53 000/830/545/0	18.8 k	38.40	2.04
文献[15]	U250	2^{16}	16	48	300	2 082/11 542/420/15	23.5 k	54.41	2.31
文献[16]	ZCU102	2^{16}	16	60	200	11 850/21 645/180/25	13.6 k	46.87	3.44
本文	ZU9EG	2^{16}	16	60	252	33 128/10 110/150/0	11.6 k	60.42	5.21

注:器件型号:U250:UltraScale Alveo U250;KCU105:Kintex UltraScale KCU105;ZU9EG:Zynq UltraScale+ XCZU9EG;ZCU102:Zynq UltraScale+ ZCU102. #Slice:#Slice ≈ LUT/4(7 series) ≈ LUT/8(UltraScale),1 DPS ≈ 102.4Slice(7 series) ≈ 51.2Slice(UltraScale),36 Kb BRAM ≈ 232.4 Slice(7 series) ≈ 116.2 Slice(UltraScale).[14,17,18]

行数据缓冲完成整体NTT运算,乘法资源的利用率不高,相比于文献[14,15],本文设计的TFG为Radix-16 NTT单元中每个stage提供对应的旋转因子,整体的TPS提升了一倍以上. 文献[16]的研究中,同样采用了一种低成本的模乘单元,但支持的模仅有12个,而本文通过深入分析,筛选并使用了120个素数模,并基于此设计了模乘单元,针对RNS-CKKS方案能够支持更多的乘法层数.

表3 ASIC实验结果对比

对比文献	工艺	设计	Radix	$\log_2(q)$	Freq	Gate/M	Area/mm^2	Cycle/M	TPG	TPA
文献[19]	90 nm CMOS	NTT	16	64	370	23	—	0.139	0.49	—
文献[20]	12 nm CMOS	NTT	256	32	2000	—	16	0.022	—	11.88
文献[21]	40 nm CMOS	NTT	16	64	500	6.55	4.25	0.018 6	17.25	26.56
	90 nm CMOS	TFG			200	1.45	4.12	0.018 6	31	10.96
本文	40 nm CMOS	NTT	16	60	571	1.89	3.40	0.016 4	72.44	40.27
		TFG	16	60	571	0.60	1.09	0.016 4	228.18	125.60

注:TPA: Throughput per square millimeter;TPG:Throughput per gate.

表3中展示了在ASIC的实验条件下,本文的设计与其他相关工作的比较,为了更好地与近几年的相关研究进行比对,我们基于第4节中低成本模乘单元的设计实现了图8所示的NTT的整体架构. 文献[19]采取乒乓存储结构,因此消耗了大量硬件资源,在TFG设计上,该文献存储单个旋转因子,并通过校正因子递归计算所有所需的旋转因子,导致硬件吞吐量有所下降,与该文献相比,本文TPG值得到了显著提高. 在与文献[20]的对比过程中,为了进行更加合理的比较,表3展示的文献[20]的TPA值是经过40 nm工艺与12 nm工艺的比例因子归一化后计算的结果,与该文献相比,本文的设计的TPA值提高了4倍. 文献[21]的设计思路与本文相近,在TFG的设计中同样使用了15个ModMult单元,除了ModMult单元之外,在该文献中,自身旋转因子的生成逻辑更加复杂,导致TFG中还消耗了大量的乘法资源,增加了硬件资源开销,本文基于合理的旋转因子生成方案和低成本的模乘单元设计,因此NTT及TFG的设计相较于文献[21]具有更好的性能表现,其中,NTT整体的TPG值提升了接近5倍,TFG设计的TPG值提升约7倍.

7 结论

本文面向全同态加密中NTT的应用需求,提出了一种动态旋转因子生成方案,基于此设计并实现了一款低成本的旋转因子发生器,降低了旋转因子的存储消耗,减小了旋转因子的传输对系统整体速率的影响,同时,设计的轻量级模乘算法也为全同态加密中模乘的设计提供了合理的设计建议.

参考文献

[1] GENTRY C. Fully homomorphic encryption using ideal lattices[C]//Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. New York: ACM, 2009: 169-178.

[2] FAN J F, VERCAUTEREN F. Somewhat practical fully homomorphic encryption[J]. IACR Cryptol. EPrint Arch., 2012, 2012: 144.

[3] BRAKERSKI Z, GENTRY C, VAIKUNTANATHAN V. (Leveled) fully homomorphic encryption without bootstrapping[C]//Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. New York: ACM, 2012: 309-325.

[4] CHEON J H, KIM A, KIM M, et al. Homomorphic encryption for arithmetic of approximate numbers[C]//Advances in Cryptology-ASIACRYPT 2017. Cham: Springer, 2017: 409-437.
[5] GENTRY C, SAHAI A, WATERS B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based[C]//Advances in Cryptology-CRYPTO 2013. Berlin: Springer, 2013: 75-92.
[6] DUCAS L, MICCIANCIO D. FHEW: Bootstrapping homomorphic eencryption in less than a second[M]//Advances in Cryptology—EUROCRYPT 2015. Berlin, Heidelberg: Springer, 2015: 617-640.
[7] 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.
[8] KIM S, KIM J, KIM M J, et al. BTS: An accelerator for bootstrappable fully homomorphic encryption[C]//Proceedings of the 49th Annual International Symposium on Computer Architecture. New York: ACM, 2022: 711-725.
[9] AGRAWAL R, DE CASTRO L, YANG G W, et al. FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption[C]//2023 IEEE International Symposium on High-Performance Computer Architecture. Piscataway: IEEE, 2023: 882-895.
[10] ZHANG N, YANG B H, CHEN C, et al. Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020: 49-72.
[11] CHEON J H, HAN K, KIM A, et al. A full RNS variant of approximate homomorphic encryption[C]//Selected Areas in Cryptography-SAC 2018. Cham: Springer, 2019: 347-368.
[12] KILTZ E, LYUBASHEVSKY V, SCHAFFNER C. A concrete treatment of fiat-Shamir signatures in the quantum random-oracle model[C]//Advances in Cryptology-EUROCRYPT 2018. Cham: Springer, 2018: 552-586.
[13] MARETA R, SATRIAWAN A, DUONG P N, et al. A bootstrapping-capable configurable NTT architecture for fully homomorphic encryption[J]. IEEE Access, 2024, 12: 52911-52921.
[14] IM N, YANG H, EOM Y, et al. Efficient twiddle factor generators for NTT[J]. Electronics, 2024, 13(16): 3128.
[15] LEE C, LEE H, DUONG-NGOC P, et al. Twiddle factor generator architecture for number theoretic transform[C]//2023 20th International SoC Design Conference. Piscataway: IEEE, 2024: 209-210.
[16] DUONG-NGOC P, KWON S, YOO D, et al. Area-efficient number theoretic transform architecture for homomorphic encryption[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2023, 70(3): 1270-1283.
[17] PATYK T, QURESHI F, TAKALA J. Hardware-efficient twiddle factor generator for mixed radix-2/3/4/5 FFTs[C]//2016 IEEE International Workshop on Signal Processing Systems. Piscataway: IEEE, 2016: 201-206.
[18] LIU W Q, FAN S L, KHALID A, et al. Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019, 27(10): 2459-2463.
[19] FENG X, LI S G. Accelerating an FHE integer multiplier using negative wrapped convolution and Ping-pong FFT[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2019, 66(1): 121-125.
[20] GEELEN R, VAN BEIRENDONCK M, PEREIRA H V L, et al. BASALISC: Programmable hardware accelerator for BGV fully homomorphic encryption[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023: 32-57.
[21] JHENG C S, WU Y T, SHIEH M D. On-the-fly twiddle factor generator design for efficient memory management of negative wrapped convolution[C]//2024 International VLSI Symposium on Technology, Systems and Applications. Piscataway: IEEE, 2024: 1-4.

作者简介

付秋兴　男,1999 年 8 月出生于河南省商丘市. 现为信息工程大学网络空间安全专业博士研究生. 主要研究方向为全同态密码处理器设计.
E-mail: 1415505333@qq.com

李　伟　男,1983 年 11 月出生于天津市. 现为信息工程大学教授. 主要研究方向为体系结构、安全芯片设计、集成电路技术.
E-mail: try_1118@163.com