doi: 10.3969/j.issn.0490-6756.2019.04.013

基于 MGSW15 方案的分组密码电路的同态运算

刘 帅, 胡 斌
(战略支援部队信息工程大学, 郑州 450001)

摘 要: 全同态加密(FHE)允许在不知道秘密信息的前提下对密文进行任意运算,已成为大数据和云安全背景下的热门研究方向,近年来取得了重大进展.但在实际应用中全同态加密仍面临诸多问题,其中严重的密文扩张给密文传输带来了巨大压力,通过将全同态加密方案与对称密码相融合可以有效解决这一问题.GSW 类型的全同态加密方案效率较高,且进行同态计算不需要再线性化技术,本文选取了支持并行操作的 MGSW15 方案,其密文可以转化为任意基于 LWE 的 FHE 方案的密文.给出了在云计算背景下基于 MGSW15 方案实现密文压缩的基本框架,并利用该方案分别同态计算实现了分组密码 AES-128、PRINCE、SIMON-64/128 电路,根据每种分组密码的结构特点对其明文分组采用多种切割方式以提高同态运算效率,最后对效率和安全性进行了分析.结合 AES 算法的安全性、通用性以及轻量级分组密码算法 PRINCE 和 SIMON 的高效性,本文的工作在实际应用中效率更高、应用范围更广,密文传输量与明文规模的比值趋近于 1,且传输 1 比特明文只需进行 $O(1)$ 次同态乘法.

关键词: 全同态加密; MGSW15 方案; AES 算法; 轻量级分组密码
中图分类号: TP309.7 **文献标识码**: A **文章编号**: 0490-6756(2019)-04-0661-10

Homomorphic operation of block cipher circuits based on MGSW15 scheme

LIU Shuai, *HU Bin*
(PLA Strategic Support Force Information Engineering University, Zhengzhou 450001, China)

Abstract: Fully homomorphic encryption (FHE), which allows arbitrary computation of ciphertexts without knowing the secret information, has become a hot research direction in the context of big data and cloud security and has made significant progress in recent years. However, in practical applications, fully homomorphic encryption still faces many problems such as the serious ciphertext expansion brings great pressure to the ciphertext transmission. This problem can be effectively solved by combining the full homomorphic encryption scheme with symmetric cipher. The GSW-type full homomorphic encryption scheme is more efficien and does not need the re-linearization technique in homomorphic operations. This paper selects the MGSW15 scheme which supports parallel operations and its ciphertexts can be transformed into the ciphertext of any LWE-based FHE scheme. We present the basic framework of ciphertext compression based on MGSW15 scheme in the cloud computing context, which implement homomorphic operations of the block cipher circuits for AES-128, PRINCE and SIMON-64/128 respectively. According to the structural characteristics of each block cipher, we use various kinds of plaintext-slicing ways to improve the efficiency of homomorphic evaluation. Finally, we analyze efficiency and secur-

收稿日期: 2018-10-15
基金项目: 国家自然科学基金(61601515); 河南省自然科学基金(162300410332)
作者简介: 刘帅(1993—), 男, 山东泰安人, 硕士, 研究方向为全同态密码. E-mail: sssshuai1993@163.com

ity of the proposed method. Combined with the security and generality of AES as well as the efficiency of lightweight block cipher algorithms PRINCE and SIMON, our work is more efficient and has a wider application range in practical applications, in which the ciphertexts communication complexity is approximately equal to the plaintexts scale and only $O(1)$ homomorphic multiplications are needed for every plaintext bit.

Keywords: Fully homomorphic encryption; MGSW15; AES; Lightweight block cipher

1 引 言

全同态加密(Fully Homomorphic Encryption,FHE)是一类公钥加密方案,允许在不知道私钥的情况下,对密文进行任意运算后解密得到的结果相当于对明文进行相应的运算.这一技术从根本上解决了将数据及操作委托给第三方时的隐私保密问题.2009 年,Gentry 基于理想格提出了第一个真正意义上的全同态加密方案[1],拉开了全同态加密研究的帷幕,之后研究者们不断提出新的方案和技术[2-7],大大提高了全同态加密的效率.2013 年 Gentry 等人[7]基于 LWE 问题和近似特征向量技术提出了 GSW13 方案,密文的同态乘法不再需要再线性化技术,减少了公钥规模,并据此构建了第一个基于身份的 FHE 方案;2015 年,Hiromasa 等人[8]提出了该方案的一个变体——MGSW15 方案,其明文为矩阵形式,实现了密文打包(将多比特明文加密为一个密文).

近年来,全同态加密技术虽不断被提高与完善,但在实际应用中仍面临诸多问题,如严重的密文扩张造成密文传输困难.在典型的全同态加密应用中,用户(Alice)首先要用 FHE 方案加密明文数据 m,将得到的密文 c 发送到云端(Cloud).由于现有的 FHE 方案密文扩张严重(加密 1 bit 明文得到的密文是 n bit,$n\gg1$),密文 c 的规模远大于明文 m,给数据传输带来很大负担,如利用 DGHV 方案加密 4 MB 明文数据得到的密文规模是 73 TB[9].利用密文打包技术[10,11]可以压缩密文规模,DGHV 方案打包加密 4 MB 明文后密文规模降到了 280 GB,但这实际上仍是不可行的.

2011 年,Naehrig 等[12]针对该问题提出了一种新的解决方法:用户 A 用对称加密方案 E 加密明文数据 m,用同态加密方案 HE 加密 E 的密钥 k,将较小的密文 $c'=(HE_{pk}(k),E_k(m))$ 发送到云端,云端得到 c' 后进行密文解压缩,即同态计算方案 E 的解密电路,得到 $c=HE_{pk}(m)$.可看作这是一种密文压缩方法,但需要说明的是,这不是真正意义上的密文压缩.显然,在这种方法中,随着明文 m 规模的增长,密文 c' 与明文 m 的规模比例趋近于 1,从信息论的角度来看,这已经达到最优,所以关键的问题是如何高效地同态计算对称加密方案 E 的解密电路.在保证安全性的前提下,如何选取对称加密方案 E 或者设计结构适合的 E,以提高密文解压缩过程的实现效率,已成为当前的一个热点问题.Naehrig 等虽然提出了"密文压缩"的基本思路,但对于结合具体 FHE 方案如何高效地实现这一技术并没有进行深入研究,在他们工作的基础上,其他学者给出了一些高效的实现方案.

值得注意的是,高效地实现"密文压缩"与全同态加密方案的设计往往是同步的:2012 年,Gentry 等[13]基于 RLWE-BGV 方案,利用 Helib 软件库实现了对 AES 电路的同态计算;2013 年,Cheon 等[11]基于 DGHV 方案设计了密文打包和同态槽移位操作,在此基础上实现了对 AES 电路的同态计算;2016 年,Canteaut 等[14]设计了流密码算法 Trivium 的改进算法 Kreyvium,基于 BGV 方案和 FV 方案分别实现了对流密码算法 Trivium,Kreyvium 和 LowMC 的同态计算.

这些方案或是基于 RLWE 问题,或是基于整数上的困难问题,仍没有公开文献专门针对基于 LWE 的 FHE 方案如何高效地实现密文压缩进行研究,我们很自然地想到 GSW 类型的全同态加密方案,也是当前基于 LWE 问题效率最高的方案.

对于对称加密方案的选择,我们分别考虑 AES 密码算法和轻量级分组密码算法.轻量级密码是针对物联网中 RFID 等应用发展起来的一类密码算法,能够在计算资源受限的情况下快速执行且保证相对的安全性.与传统密码算法相比,轻量级密码算法结构简单,实现效率高.2012 年,Borghoff 等[15]提出了轻量级分组密码算法 PRINCE;2013 年 6 月美国国家安全局提出了一种轻量级分组密码 SIMON[16].

本文基于 MGSW15 方案,给出了"密文压缩"过程的基本框架,分别实现了分组密码 AES-128、PRINCE 和 SIMON-64/128 电路的同态计算,并就效率和安全性进行了分析比较.由于不需要再线

性化技术和 bootstrapping 技术，本文方案实现效率更高，且云端可以将得到的密文转化为任意 LWE 上 FHE 方案的密文，使得本文的方案应用范围更广.

2　基础知识

本文中，Z 表示整数集合，$a \xleftarrow{U} G$ 表示 a 从集合 G 中均匀随机抽样，$b \xleftarrow{R} p$ 表示 b 从概率分布 P 中抽样. 向量用粗体小写字母表示，如 x，且默认为列向量，其转置表示为 x^T，两个向量的内积用 $<x,y>$ 表示；矩阵用粗体大写字母表示，如 X，矩阵的第 i 列向量用 x_i 表示，第 i 行第 j 列元素用 $X[i,j]$ 表示，矩阵的转置用 X^T 表示，$[X \| Y]$ 表示两个矩阵的级联，I_n 表示 n 阶单位矩阵. 对于 $k \in \mathbf{Z}$，记集合 $[k]=\{1,2,\cdots,k\}$.

2.1　全同态加密方案 MGSW15

GSW 类型的 FHE 方案（如文献[7]中的 GSW13 方案）是基于近似特征向量构造的新型全同态加密方案，其密文为矩阵形式. 2015 年，Hiromasa 等人[8]提出了该类方案的一个变体——MGSW15 方案，允许对矩阵加密，实现了密文打包. 该方案由以下 5 个算法组成.

(1) 初始化算法 $Setup$. 输入安全参数 l，根据 l 决定格的维数 n、模数 q 以及 Z 上的亚高斯分布 c. 令 $l=\lceil \log q \rceil, m=O((n+r)\log q), N=(n+r)\cdot l$，$r$ 是待加密的方阵维数，定义了明文空间 $\{0,1\}^{r\times r}$，密文空间是 $Z_q^{(n+r)\times N}$. 令矩阵 $G=g^T \otimes I_{n+r}$，$g^T=(1,2,\cdots,2^{l-1})$，输出参数 n,q,m,l,N,c.

(2) 密钥生成算法 KeyGen$(1^l,r)$. 均匀随机抽样矩阵 $A \xleftarrow{U} Z_q^{n\times m}$，密钥矩阵 $S' \xleftarrow{R} c^{r\times n}$，噪声矩阵 $E \xleftarrow{R} c^{r\times m}$. 令 $S=[I_r \| -S']\in \mathbf{Z}_q^{r\times(n+r)}$，计算 $B=\left(\frac{S'A+E}{A}\right)\in \mathbf{Z}_q^{(n+r)\times m}$. 令 $M_{(i,j)}\in\{0,1\}^{r\times r}$，$(i,j=1,\cdots,r)$ 表示第 (i,j) 个元素为 1，其余元素为 0 的矩阵. 对所有的 $i,j=1,\cdots,r$，抽样 $R_{(i,j)} \xleftarrow{U} \{0,1\}^{m\times N}$，计算 $P_{(i,j)}=BR_{(i,j)}+\left(\frac{M_{(i,j)}S}{0}\right)G \in \mathbf{Z}_q^{(n+r)\times N}$，输出 $pk=(\{P_{(i,j)}\}_{i,j\in[r]},B)$ 和 $sk=S$.

(3) 加密算法 $Enc_{pk}(M\in\{0,1\}^{r\times r})$. 随机抽样矩阵 $R \xleftarrow{U} \{0,1\}^{m\times N}$，输出密文 $C=BR+\sum_{i,j\in[r];M[i,j]=1} P_{(i,j)} \in \mathbf{Z}_q^{(n+r)\times N}$，其中，$M[i,j]$ 是 m 的第 (i,j) 个元素.

(4) 解密算法 $\mathrm{Dec}_{sk}(C)$. 输出矩阵 $M=(\lfloor <s_i^T, c_{jl-1}> \rceil_2)_{i,j\in[r]}\in\{0,1\}^{r\times r}$.（其中，$s_i^T$ 表示 S 的第 i 行，c_{jl-1} 表示 C 的第 $jl-1$ 列. 对于 $x\in\mathbf{Z}_q$，如果 x 接近 $q/4$，函数 $\lfloor x \rceil_2$ 输出 1，否则输出 0.）

(5) 同态运算. 同态加法 MGSW. Add(C_1,C_2)：输出 $C_{\mathrm{add}}=C_1+C_2\in\mathbf{Z}_q^{(n+r)\times N}$；同态乘法 MGSW. Mult$(C_1,C_2)$：输出 $C_{\mathrm{mult}}=C_1G^{-1}(C_2)\in\mathbf{Z}_q^{(n+r)\times N}$，其中 G^{-1} 是一个随机函数[17]：对于矩阵 $C\in\mathbf{Z}_q^{(n+r)\times N}$，$G^{-1}(C)$ 输出一个矩阵 $X'\in\mathbf{Z}_q^{N\times N}$，满足 $GX'\equiv C(\mathrm{mod}\ q)$.

该方案密文的同态加法和同态乘法分别对应着明文矩阵的加法和乘法.

2.2　基于 MGSW15 方案实现并行运算

2.1 节所述的 MGSW15 方案实现了矩阵加法和乘法的同态计算，文献[8]给出了基于该方案实现并行运算的方法，具体细节介绍如下.

在 MGSW15 方案中，假设两个明文矩阵 M、M' 以及它们对应的密文 C、C'，将 M 中的每个元素 $M[i,j]$ 称作一个明文槽. 密文 C、C' 的同态加法和乘法对应着明文矩阵 M、M' 的加法和乘法，对单个明文槽无法进行同态运算. 令 M、M' 为对角矩阵，M、M' 的加法和乘法就对应着对角线元素的加法和乘法，这样就可以通过密文的同态加法和乘法实现明文槽间的并行同态加法和乘法运算. 为了实现矩阵 M 第 i 个对角线元素与矩阵 M' 第 j 个对角线元素的同态加法或乘法 $(i\neq j)$，需要对明文槽进行同态置换运算，通过在密文左右分别乘上该置换对应的置换矩阵的加密和它的逆矩阵的加密可以实现对角线上明文槽的置换，算法如下.

(1) $-$SwitchKeyGen(pk,σ)：已知公钥 pk 和置换 σ，令 $\Sigma\in\{0,1\}^{r\times r}$ 是对应于置换 σ 的矩阵，计算 $W_\sigma \xleftarrow{R} \mathrm{Enc}_{pk}(\Sigma)$，$W_{\sigma^{-1}} \xleftarrow{R} \mathrm{Enc}_{pk}(\Sigma^{-1})$，输出转换密钥 $\mathrm{ssk}_\sigma=(W_\sigma,W_{\sigma^{-1}})$.

(2) $-\mathrm{SlotSwitch}_{\mathrm{ssk}_\sigma}(C)$：输入转换密钥 ssk_σ 和密文 C，输出 $C_\sigma \xleftarrow{R} W_\sigma \odot C \odot W_{\sigma^{-1}}$，这里 $\odot$ 表示同态乘法.

由以上算法可以实现明文矩阵中对角线元素的任意置换.

2.3　轻量级分组密码 PRINCE

PRINCE 算法是一种轻量级分组密码算法，采用了 S-P 网络结构，其分组长度为 64 比特，密钥长度为 128 比特，迭代圈数为 12. 待加密的 64 比特明文分组用状态矩阵（4×4 的半字节矩阵）表示，如 $(\alpha_i(i=0,1,\cdots,15)$ 代表半个字节）.

$$\begin{pmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 \\ \alpha_8 & \alpha_9 & \alpha_{10} & \alpha_{11} \\ \alpha_{12} & \alpha_{13} & \alpha_{14} & \alpha_{15} \end{pmatrix}$$

圈函数的输入是 4×4 的半字节矩阵,每一圈包括四层变换:4 比特 S 盒,线性变换,圈常数加法和圈密钥加法,加密过程如图 1 所示.

PRINCE 算法的具体细节介绍如下.

(1) 密钥生成:PRINCE 算法的 128 比特密钥分成两部分 k_0 和 k_1,令 $k_0'=(k_0>>>1)\oplus(k_0>>63)$,其中,$k_0$ 和 k_0' 是白化密钥,k_1 是所有圈共同的圈密钥. 另外 PRINCE 算法定义了 12 个 64 比特的圈常数 RC_i, $(i=0,1,\cdots,11)$.

(2) S 盒:4 进 4 出的 S 盒是该算法中唯一的非线性操作,定义如表 1.

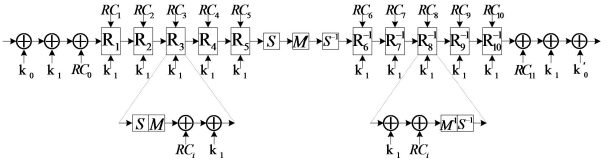

图 1 PRINCE 算法的加密过程

Fig. 1 Encryption process of PRINCE

表 1 PRINCE 算法的 S 盒

Tab. 1 Sbox of PRINCE

i	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[i]	B	F	3	2	A	C	9	1	6	7	8	0	E	5	D	4

(3) 线性变换 M/M':M' 是由一个 64×64 矩阵 M' 定义的线性变换,具体细节可参考文献[15],它使得每一个输出比特都是 3 个来自不同半字节的输入比特之和,且 M' 是对合矩阵. 线性变换 m 包括两部分:行移位变换和线性变换 M',其中行移位变换与 AES 完全相同.

3 基于 MGSW15 方案实现密文压缩

我们分析了现有的 FHE 方案的特点,最终选取 MGSW15 方案实现密文压缩,该方案具有以下优点:(1) 噪声增长速度慢;(2) 公钥规模小,不需要 Evaluate 公钥,减少了数据传输量;(3) 同态乘法后无需"再线性化"操作,同态运算速度更快;(4) MGSW15 方案的密文可以转化为任意基于 LWE 的 FHE 方案的密文;(5) 允许密文打包,支持并行操作. 本节主要介绍基于 MGSW15 方案实现密文压缩的基本框架,本文规定 MGSW15 方案的明文空间为 r 阶对角矩阵.

在全同态加密的实际应用中,用户 A 利用 FHE 方案加密明文数据 m 后传输给云端,为减少数据传输量,我们在 Naehrig 等[12]工作的基础上,基于 MGSW15 方案实现"密文压缩". 首先用户 A 初始化密钥参数:对称加密方案 E 的密钥 k,MGSW15 方案的公私钥对 (pk,sk). 利用对称加密方案 E 加密数据 m 得到 $E_k(m)$,利用 MGSW15 方案加密方案 E 的密钥 k 得到 $MGSW.\mathrm{Enc}_{pk}(k)$($k$ 按一定顺序排列在明文矩阵的对角线上),将密文 $c'=(E_k(m),\mathrm{MGSW.Enc}_{pk}(k))$ 发送到云端,云端利用 MGSW15 方案的同态性质解压缩 c' 得到密文.

$$C=\mathrm{MGSW.Enc}_{pk}(m)=\mathrm{Evaluate}(C_{E^{-1}},(\mathrm{MGSW.Enc}_{pk}(E_k(m)),\mathrm{MGSW.Enc}_{pk}(k)))\in\mathbf{Z}_q^{(n+r)\times N} \quad (1)$$

最后,得到了 MGSW15 方案的密文 $C\in\mathbf{Z}_q^{(n+r)\times N}$,对应着明文矩阵 $M\in\{0,1\}^{r\times r}$ 和密钥 $S=[I_r\parallel -S']\in\mathbf{Z}_q^{r\times(n+r)}$,其中 $S'\xleftarrow{R}c^{r\times n}$,解密方程为 $M=(\lfloor<s_i^T,c_{jl-1}>\rceil_2)_{i,j\in[r]}$. 目前已知的 LWE 上的 FHE 方案的解密方程都可以表示成 $m=\lfloor<s,c>\rceil_2$,其中,S、C 分别是密钥向量和密文向量,所以云端可以将密文 C 转化为任意 LWE 上的 FHE 方案的密文[8]:只取密文 C 的第 $il-1$ 列 c_{il-1},便可以得到文献[2-4]中方案的密文形式,其对应的密钥为 S 的第 i 行 s_i^T,明文为 m 中第 i 个对角线元素 $M[i,i]$,解密方程为 $m=\lfloor<s_i^T,c_{il-1}>\rceil_2$;保留密文 C 的第 i 行和后 n 行,第 $(i-1)l+1$ 列到第 il 列以及后 nl 列,删去其他行和列就可以得到 GSW13 方案[7]的密文形式,其对应的密钥为 S 的第 i 行 s_i^T(只取第 i 个元素和后 n 个元素),明文为 m 中第 i 个对角线元素 $M[i,i]$,解密方程为 $m=\lfloor<s_i^T,c_{il-1}>\rceil_2$. 这增加了应用的灵活性,云端不仅能得到 MGSW15 方案的密文,通过变换还能得到任意 LWE 上 FHE 方案的密文,之后云端可以基于对应的 FHE 方案对得到的密文进行其它同态操作,如云计算、隐私保护检索等.

第 4、5 节我们将具体的分组密码算法 AES、PRINCE 和 SIMON 应用到(1)式中,且通过并行

操作提高(1)式的实现效率. AES 作为高级加密标准，早已广泛应用，其集安全、性能、效率、可实现性及灵活性于一身，我们选择 AES 主要基于其以下优点：(1) 能抵抗目前已知的所有攻击；(2) 能在各种平台上快速有效地实现；(3) 规则的代数结构使其具有良好的并行实现潜力. 相比较于 AES 等传统密码算法，轻量级密码在牺牲了一定安全性的同时具有实现速度快的优点，其中，PRINCE、SIMON 算法结构简单、延时低，能够实现高效的同态计算，当实际应用中对安全性要求不高时，可以采用 PRINCE、SIMON 算法.

这三个分组密码算法中每个算法的加密与解密过程相似，我们只给出每个算法加密电路的同态计算过程，解密电路的同态计算方法类似.

4　AES-128 电路的同态计算

这一部分我们基于 MGSW15 方案同态计算 AES-128 电路. AES-128 属于迭代型分组密码算法，其分组长度为 128 比特，密钥长度为 128 比特，迭代圈数为 10. 待加密的 128 比特明文分组和 128 比特圈密钥用状态矩阵(4×4 的字节矩阵)表示，如(α_i($i=0,1,\cdots,15$)代表一个字节).

$$\begin{pmatrix}\alpha_0 & \alpha_1 & \alpha_2 & \alpha_3\\ \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7\\ \alpha_8 & \alpha_9 & \alpha_{10} & \alpha_{11}\\ \alpha_{12} & \alpha_{13} & \alpha_{14} & \alpha_{15}\end{pmatrix}$$

圈函数的输入是 4×4 的字节矩阵，每一圈包括四层变换，依次是字节代替变换，行移位变换，列混合变换和圈密钥加法，最后一圈没有列混合变换.

同 2.2 节所述，为了实现并行运算，令 MGSW15 方案的明文空间为 r 阶对角矩阵，对角线元素对应 r 个明文槽. 下面分两种方法同态计算 AES-128 电路，分别是逐字节切割和逐状态切割.

4.1　AES-128 电路的逐字节切割同态计算

首先介绍逐字节切割，用一个 16×8 矩阵表示 AES 状态，矩阵的每一行对应一个字节的 8 个比特. 在用 MGSW15 方案对明文状态加密时，第 i 列 16 个比特按顺序排放在明文槽 $M_i[j,j]$，($i=0,1,\cdots,7$)中，即 $M_i[l\cdot 16+j, l\cdot 16+j]$是第 l 个 AES 状态的第 j 个字节中的第 i 比特，经过 MGSW15 方案加密后得到 8 个密文 $C_0,C_1,\cdots,C_7$. 这样 AES 状态包括密文 $C_0,C_1,\cdots,C_7$，我们称其为 HAES 状态，下文的同态操作都是对密文进行的，可以并行处理 $k=\lfloor r/16\rfloor$ 个 AES 加密过程. 同样，每一圈的圈密钥用相同的形式进行加密，注意圈密钥的每一比特要重复 k 次，记第 t 圈圈密钥对应的密文为 $C_{t,0},C_{t,1},\cdots,C_{t,7}$. 下面详细给出同态计算 AES-128 圈函数中的每个变换：

(1) 字节代替变换. 利用 S 盒对状态矩阵中每个字节进行代替变换，为了简化 S 盒电路，尽可能地减少乘法运算，我们采用了 Boyar 和 Peralta 的优化技术[18]，用简化的布尔电路表示 S 盒，共进行 32 次乘法运算、83 次加法运算，这是目前已知的对 AES-S 盒的最小电路表示. 对密文 $C_0,C_1,\cdots,C_7$ 进行相应操作即可，这样可以并行地计算所有字节. 这一步骤需要 83 次MGSW. Add和 32 次 MGSW. Mult.

(2) 行移位变换状态矩阵的第 i 行 4 个字节循环左移 i 位，我们通过 2.2 节中介绍的同态置换操作完成，具体过程如下，令 r 元置换

$$\sigma=\begin{pmatrix}0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15\\ 0 & 1 & 2 & 3 & 5 & 6 & 7 & 4 & 10 & 11 & 8 & 9 & 15 & 12 & 13 & 14\end{pmatrix}\cdots\cdots$$

每个密文有 r 个明文槽，这里只列出了置换中前 16 个元素的对应关系，省略部分遵循同样规则，本文后面提到的置换都是如此. 计算

$$\mathrm{ssk}_\sigma \xleftarrow{R} \mathrm{SwitchKeyGen}(pk,\sigma)$$

$$C_i \xleftarrow{R} \mathrm{SlotSwitch}_{\mathrm{ssk}_\sigma}(C_i),\ (i=0,1,\cdots,7)$$

这一步骤需要 16 次 MGSW. Mult.

(3) 列混合变换. 把 4×4 状态矩阵左乘一个矩阵，运算在 $GF(2^8)$上进行，过程如下.

$$\begin{pmatrix}\alpha'_0 & \alpha'_1 & \alpha'_2 & \alpha'_3\\ \alpha'_4 & \alpha'_5 & \alpha'_6 & \alpha'_7\\ \alpha'_8 & \alpha'_9 & \alpha'_{10} & \alpha'_{11}\\ \alpha'_{12} & \alpha'_{13} & \alpha'_{14} & \alpha'_{15}\end{pmatrix}=\begin{pmatrix}0x02 & 0x03 & 0x01 & 0x01\\ 0x01 & 0x02 & 0x03 & 0x01\\ 0x01 & 0x01 & 0x02 & 0x03\\ 0x03 & 0x01 & 0x01 & 0x02\end{pmatrix}\begin{pmatrix}\alpha_0 & \alpha_1 & \alpha_2 & \alpha_3\\ \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7\\ \alpha_8 & \alpha_9 & \alpha_{10} & \alpha_{11}\\ \alpha_{12} & \alpha_{13} & \alpha_{14} & \alpha_{15}\end{pmatrix}$$

可以把整个过程表示为

$$\begin{pmatrix} \alpha_0' & \alpha_1' & \alpha_2' & \alpha_3' \\ \alpha_4' & \alpha_5' & \alpha_6' & \alpha_7' \\ \alpha_8' & \alpha_9' & \alpha_{10}' & \alpha_{11}' \\ \alpha_{12}' & \alpha_{13}' & \alpha_{14}' & \alpha_{15}' \end{pmatrix} = 0x02 \times \begin{pmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 \\ \alpha_8 & \alpha_9 & \alpha_{10} & \alpha_{11} \\ \alpha_{12} & \alpha_{13} & \alpha_{14} & \alpha_{15} \end{pmatrix} + 0x03 \times$$

$$\begin{pmatrix} \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 \\ \alpha_8 & \alpha_9 & \alpha_{10} & \alpha_{11} \\ \alpha_{12} & \alpha_{13} & \alpha_{14} & \alpha_{15} \\ \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \end{pmatrix} + \begin{pmatrix} \alpha_8 & \alpha_9 & \alpha_{10} & \alpha_{11} \\ \alpha_{12} & \alpha_{13} & \alpha_{14} & \alpha_{15} \\ \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 \end{pmatrix} + \begin{pmatrix} \alpha_{12} & \alpha_{13} & \alpha_{14} & \alpha_{15} \\ \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 \\ \alpha_8 & \alpha_9 & \alpha_{10} & \alpha_{11} \end{pmatrix} \tag{2}$$

所以,我们定义了另外 3 个 HAES 状态(24 个密文)$C_i^1, C_i^2, C_i^3 (i=0,1,\cdots,7)$,由输入的 HAES 状态置换得到:令 r 元置换

$$\sigma_1 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \cdots\cdots \\ 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 0 & 1 & 2 & 3 \end{pmatrix}$$

$$\sigma_2 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \cdots\cdots \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}$$

$$\sigma_3 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \cdots\cdots \\ 12 & 13 & 14 & 15 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{pmatrix}$$

计算 $\mathrm{ssk}_{\sigma_j} \xleftarrow{R} \mathrm{SwitchKeyGen}(pk, \sigma_j), (j=1,2,3)$;$C_i^j \xleftarrow{R} \mathrm{SlotSwitch}_{\mathrm{ssk}_{\sigma_j}}(C_i), (i=0,1,\cdots,7,\ j=1,2,3)$;$GF(2^8)$ 上的乘法 $b'=0x02\times b, c'=0x03\times c (b,c\in GF(2^8))$ 可转化为 $GF(2)$ 上的运算,见算法 1 和算法 2.

算法 1 输入 $b=(b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0)\in GF(2^8)$,计算 $b_0'=b_7, b_1'=b_0\oplus b_7, b_2'=b_1, b_3'=b_2\oplus b_7, b_4'=b_3\oplus b_7, b_5'=b_4, b_6'=b_5, b_7'=b_6$. 输出 $b'=0x02\times b=(b_7',b_6',b_5',b_4',b_3',b_2',b_1',b_0')$.

算法 2 输入 $c\in GF(2^8)$,调用算法 1 计算 $c'=0x02\times c+c$,输出 c'.

对 HAES 状态应用算法 1 和算法 2,可以并行地计算所有 k 个 AES 状态的所有字节,具体操作为:对 C_i、C_i^1 分别应用算法 1 和算法 2 得到新的 C_i 与 C_i^1,只需要 3+3+8=14 个 MGSW. Add. 最后需要计算 $C_i=\mathrm{MGSW.Add}(C_i,C_i^1,C_i^2,C_i^3), (i=0,1,\cdots,7)$.

这一步骤总共需要 14+24=38 个 MGSW. Add 和 48 个 MGSW. Mult.

(4) 圈密钥加法. 把 AES 状态与圈密钥逐比特模 2 加,这一操作只需要 8 次 MGSW. Add. 如第 t 圈,只需计算如下.

$C_i=\mathrm{MGSW.Add}(C_i,C_{t,i}), (i=0,1,\cdots,7)$

其中,$C_{t,i}$ 是第 t 圈圈密钥对应的密文.

整个 AES 加密过程包括 11 个圈密钥加法,10 个字节代替变换,10 个行移位变换以及 9 个列混合变换,这个过程共需 1260 次 MGSW. Add、912 次 MGSW. Mult,可并行同态计算 $k=\lfloor r/16 \rfloor$ 个 AES 加密电路.

4.2 AES-128 电路的逐状态切割同态计算

不同于逐字节切割,逐状态切割将 AES 状态中的每一比特单独存放在不同密文中:第 l 个 AES 状态的第 i 个字节中的第 j 比特存放在 $M_{8i+j}[l,l]$ 中,经过 MGSW15 方案加密后得到 128 个密文 $C_0,C_1,\cdots,C_{127}$. 这 128 个密文组成 HAES 状态,整个 AES 加密过程只需对比特进行运算,可以并行计算 $k=r$ 个 AES 加密过程. 同样,每一圈的圈密钥用相同的形式进行加密,注意圈密钥的每一比特要重复 k 次,记第 t 圈圈密钥对应的密文为 $C_{t,0},C_{t,1},\cdots,C_{t,127}$. 下面详细介绍在这种表示下同态计算 AES-128 圈函数中每个变换的步骤.

(1) 字节代替变换. 同样采用 Boyar 和 Peralta 优化的 S 盒电路,这里需要对 AES 状态所有 16 个字节分别进行计算,所以需要 16×83=1328 次 MGSW. Add 和 16×32=512 次 MGSW. Mul.

(2) 行移位变换. 这一步骤只需要置换密文下标即可,不需要任何同态操作. 置换如下.

$$C_{8i+j} = \begin{cases} C_{8i+j}, & 0\leqslant i\leqslant 3 \\ C_{8[4+(i+1)\bmod 4]+j}, & 4\leqslant i\leqslant 7 \\ C_{8[8+(i+2)\bmod 4]+j}, & 8\leqslant i\leqslant 11 \\ C_{8[12+(i+3)\bmod 4]+j}, & 12\leqslant i\leqslant 15 \end{cases} \quad (j=0,1,\cdots,7)$$

(3) 列混合变换. 这一步与逐字节切割中的实现方式相似，同样把整个过程表示为(2)式，我们定义了另外 3 个 HAES 状态(3×128=384 个密文) $C_i^1, C_i^2, C_i^3, (i=0,1,\cdots,127)$：对照(2)式中右面三个矩阵对 HAES 状态 C_i 的下标进行置换分别得到 C_i^1, C_i^2, C_i^3. 对 C_i 中每一字节应用算法 1，对 C_i^1 中每一字节应用算法 2 得到新的 C_i 与 C_i^1，最后计算 C_i=MGSW. Add(C_i, C_i^1, C_i^2, C_i^3), $(i=0,1,\cdots,127)$，这一步骤共需 608 个 MGSW. Add.

(4) 圈密钥加法. 假设在第 t 圈，只需计算如下.

C_i=MGSW. Add($C_i, C_{t,i}$), $(i=0,1,\cdots,127)$

其中，$C_{t,i}$是第 t 圈圈密钥对应的密文，这一操作需要 128 次 MGSW. Add.

综上所述，整个过程共需 20160 次 MGSW. Add 和 5120 次 MGSW. Mult，可并行同态计算 $k=r$ 个 AES 加密电路.

5　轻量级分组密码电路的同态计算

为了实现更高效的数据加密，密码学者们提出了轻量级密码算法，如 Present, KATAN, TEA 和 HIGHT 等，文献[19]给出了这些算法的性能综述. 在这些算法中，PRINCE[15] 和 SIMON[16] 有更低的延时，为了提高密文解压缩的效率，我们尝试用 PRINCE 和 SIMON 两种轻量级分组密码算法代替 AES-128. PRINCE 算法的密钥长度也是 128 比特，我们选择密钥长度同样是 128 比特的 SIMON-64/128. 需要注意的是，轻量级密码最初是针对 RFID、无线传感器等对安全性要求不高的应用提出的，虽然这两种算法实现速度更快，但相比于 AES 等传统密码算法具有较弱的安全性，在应用中需要权衡效率与安全做出合适的选择.

下面详细介绍如何同态计算 PRINCE 和 SIMON-64/128 电路，同 2.2 节，为了实现并行计算，令 MGSW15 方案的明文空间为 r 阶对角矩阵，对角线元素对应 r 个明文槽.

5.1　PRINCE 电路的同态计算

本文在 2.3 节已经介绍了 PRINCE 算法的具体细节，针对 PRINCE 算法的特点，逐状态切割更加便于高效实现同态计算. 同 4.2 节一样，将 PRINCE 状态矩阵中的每一比特单独存放在不同密文中：第 l 个 PRINCE 状态的第 i 个半字节中的第 j 比特存放在 $M_{4i+j}[l,l]$中，经 MGSW15 方案加密后得到密文 $C_0, C_1, \cdots, C_{63}$. 这 64 个密文组成一个密文状态，整个加密过程只需对比特进行运算，可以并行计算 $k=r$ 个 PRINCE 加密过程. 同样，白化密钥 k_0 和 k_0'，圈密钥 k_1 以及圈常数 RC_i 用相同的形式进行加密，注意在这里每一比特要重复 k 次，因为同时进行了 k 次加密过程. 记 k_0、k_0'、k_1 和 RC_i 对应的密文分别为 $C_{k_0,j}, C_{k_0',j}, C_{k_1,j}, C_{RC_i,j}, (j=0,1,\cdots,63)$，下面给出同态计算 PRINCE 加密电路中每个变换的详细过程.

(1) S 盒. 根据 2.3 节中的表 1，利用软件 Mathematica 得到 S 盒的代数正规型(ANF)$(A,B,C,D)=S(a,b,c,d)$.

$A=a\oplus c\oplus ab\oplus bc\oplus abd\oplus acd\oplus bcd\oplus 1$;

$B=a\oplus d\oplus ac\oplus ad\oplus cd\oplus abc\oplus acd$;

$C=ac\oplus bc\oplus bd\oplus abc\oplus bcd\oplus 1$;

$D=a\oplus b\oplus ab\oplus ad\oplus bc\oplus cd\oplus bcd\oplus 1$.

根据上式，计算 S 盒需要 28 次乘法，可以多次利用已经计算出的中间值来减少乘法次数，具体操作如下：计算出所有二次项并存储 ab, ac, ad, bc, bd, cd，在此基础上计算出所有三次项并存储，$abc=ab\cdot c, abd=ab\cdot d, acd=a\cdot cd, bcd=b\cdot cd$. 这样 S 盒的实现只需要进行 10 次乘法，这里需要对 PRINCE 状态所有 16 个半字节分别进行计算，所以需要 16×25=400 次 MGSW. Add 和 16×10=160 次 MGSW. Mult，S 盒的逆变换同样如此.

(2) 行移位变换. 这一步骤只需要置换密文下标即可，不需要任何同态操作. 置换如下.

$$C_{4i+j}=\begin{cases} C_{4i+j}, & 0\leqslant i\leqslant 3 \\ C_{4[4+(i+1)\bmod 4]+j}, & 4\leqslant i\leqslant 7 \\ C_{4[8+(i+2)\bmod 4]+j}, & 8\leqslant i\leqslant 11 \\ C_{4[12+(i+3)\bmod 4]+j}, & 12\leqslant i\leqslant 15 \end{cases} \quad (j=0,1,2,3)$$

(3) 线性变换 M'. 由于 M'的每一个输出比特都是 3 个来自不同半字节的输入比特之和，所以这一步骤总共需要 64×2=128 个 MGSW. Add. 由于 M'是对合的，所以其逆变换也是 M'.

(4) 密钥加法/圈常数加法. 这里以圈密钥 k_1 进行举例，只需计算：C_i=MGSW. Add($C_i, C_{k_1,i}$) $(i=0,1,\cdots,63)$，其中，$C_{k_1,i}$是圈密钥 k_1 对应的密文，这一操作需要 64 次 MGSW. Add.

综上所述，整个同态计算过程总共需要 7872 次 MGSW. Add，1920 次 MGSW. Mult，可并行同态计算 $k=r$ 个 PRINCE 加密电路.

5.2　SIMON-64/128 电路的同态计算

我们基于 MGSW15 方案同态计算轻量级分组密码 SIMON-64/128 电路. 该算法采用 Feistel

结构,其分组长度为 64 比特,密钥长度为 128 比特,迭代圈数为 44,其 64 比特分组由两个 32 比特字组成,表示为(x_i,y_i),x_i,y_i 以及第 i 圈圈密钥 k_i 都是 32 比特字,第 i 圈圈函数如图 2 所示.

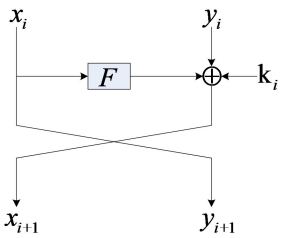

图 2 SIMON 算法的圈函数
Fig. 2 The SIMON round function

其中,函数 F 定义为 $F(x)=((x<<<8)\otimes(x<<<1))\oplus(x<<<2)$,$(x<<<j)$表示 X 循环左移 j 位;$\otimes$表示逐位模 2 乘;$\oplus$表示逐位模 2 加. 关于密钥生成等细节可以参考文献[16].

下面分两种方法同态计算 SIMON-64/128 电路,分别是逐状态切割和半状态切割.

5.2.1 SIMON-64/128 电路的逐状态切割同态计算 同上文介绍的逐状态切割一样,将 SIMON-64/128 一个分组的 64 比特按顺序加密在 64 个不同密文 $C_0,C_1,\cdots,C_{63}$ 中,更详细地,64 个密文对应的明文分别为 $M_0,M_1,\cdots,M_{63}$,$M_i[l,l]$是第 l 个 SIMON-64/128 分组的第 i 比特,这样可以并行同态计算 $k=r$ 个 SIMON-64/128 加密电路. 每一圈的圈密钥用相同的形式进行加密,注意圈密钥的每一比特要重复 k 次,记第 t 圈圈密钥对应的密文为 $C_{t,0},C_{t,1},\cdots,C_{t,31}$.

在逐状态切割表示下,同态计算过程比较简单,函数 F 中循环左移 a 位只需要改变密文下标,令 $C_j=C_{(j+a)\bmod 32}(j=0,1,\cdots,31)$,模 2 加、模 2 乘只需把对应位置的密文进行同态加法和同态乘法操作,左右字交换也只需令 $C_j\leftrightarrow C_{j+32}(j=0,1,\cdots,31)$,所以第 i 圈圈函数的同态计算如下.

$$C^*_{32+j}=C_j,C^*_j=(C_{(j+8)\bmod 32}\cdot C_{(j+1)\bmod 32})+C_{(j+2)\bmod 32}+C_{32+j}+C_{i,j},(j=0,1,\cdots,31)$$

其中,$\cdot$表示同态乘法;$+$表示同态加法. 这个过程需要 96 次 MGSW. Add 和 32 次 MGSW. Mult.

SIMON-64/128 算法迭代圈数为 44,所以整个同态计算过程总共需要 4224 次 MGSW. Add,1408 次 MGSW. Mult,可并行同态计算 $k=r$ 个 SIMON-64/128 加密电路.

5.2.2 SIMON-64/128 电路的半状态切割同态计算 我们将 SIMON-64/128 分组的左右 32 比特字分别看作一个整体进行运算,在这种表示下,将一个分组的两个 32 比特字分别加密在两个不同密文 C_0,C_1 中,对应的明文分别是 M_0,M_1,$M_0[32i+j,32i+j]$是第 i 个分组的第 j 比特,$M_1[32i+j,32i+j]$是第 i 个分组的第 $32+j$ 比特$(j=0,1,\cdots,31)$,这样可以并行同态计算 $k=\lfloor r/32\rfloor$ 个 SIMON-64/128 加密电路. 每一圈的圈密钥用相同的形式进行加密,记第 t 圈圈密钥对应的密文为 C^*_t,圈密钥的每一比特重复 k 次. 在这种表示下第 i 圈圈函数的同态计算过程如下.

函数 F 中的循环左移只需对密文 C_0 进行同态置换操作.

$$C^1_0=\mathrm{SlotSwitch}_{\mathrm{ssk}_{\sigma 1}}(C_0)$$
$$C^2_0=\mathrm{SlotSwitch}_{\mathrm{ssk}_{\sigma 2}}(C_0)$$
$$C^3_0=\mathrm{SlotSwitch}_{\mathrm{ssk}_{\sigma 3}}(C_0)$$

其中,置换 $\sigma_1,\sigma_2,\sigma_3$ 分别对应循环左移 8 位,循环左移 1 位,循环左移 2 位.

最终得到 $C'_0=(C^1_0\cdot C^2_0)+C^3_0+C_1+C^*_i$,$C'_1=C_0$. 其中,$\cdot$表示同态乘法;$+$表示同态加法.

整个同态计算过程总共需要 132 次 MGSW. Add,308 次 MGSW. Mult,可并行同态计算 $k=\lfloor r/32\rfloor$ 个 SIMON-64/128 加密电路.

6 分组密码电路同态计算的性能分析

上文中,我们在 MGSW15 方案的基础上,同态计算实现了分组密码 AES-128、PRINCE 和 SIMON-64/128 电路,下面将从效率和安全性两个方面对他们进行比较.

6.1 效率比较

在全同态加密方案中,同态乘法引入的噪声和所需的计算量都远大于同态加法,所以这里我们主要比较同态计算中同态乘法的次数,由于几种算法分组长度、并行运算组数不同,为了便于比较,我们考察在同态计算中对每比特明文加密平均需要的同态乘法次数.

AES-128 电路的同态计算中,对于逐字节切割方法,并行同态计算 $k=\lfloor r/16\rfloor$ 个 AES 加密电路总共需要 1260 次 MGSW. Add 和 912 次 MGSW. Mult,加密了 $8r$ 比特明文,平均加密每比特明文进行 $114/r$ 次 MGSW. Mult;对于逐状态切割方法,并行同态计算 $k=r$ 个 AES 加密电路需要 20160 次 MGSW. Add 和 5120 次 MGSW. Mult,

加密了 $128r$ 比特明文，平均加密每比特明文进行 $40/r$ 次 MGSW. Mult. 理论上，逐状态切割实现的效率更高.

PRINCE 电路的同态计算中，采用了逐状态切割方法，并行同态计算 $k=r$ 个 PRINCE 加密电路总共需要 7872 次 MGSW. Add 和 1920 次 MGSW. Mult，加密了 $64r$ 比特明文，平均加密每比特明文进行 $30/r$ 次 MGSW. Mult.

SIMON-64/128 电路的同态计算中，对于逐状态切割方法，并行同态计算 $k=r$ 个加密电路总共需要 4224 次 MGSW. Add 和 1408 次 MGSW. Mult，加密了 $64r$ 比特明文，平均加密每比特明文进行 $22/r$ 次 MGSW. Mult；对于半状态切割方法，并行同态计算 $k=\lfloor r/32 \rfloor$ 个加密电路需要 132 次 MGSW. Add、308 次 MGSW. Mult，加密了 $2r$ 比特明文，平均加密每比特明文进行 $154/r$ 次 MGSW. Mult.

表 2　几种分组密码电路同态计算的效率比较

Tab. 2　Efficiency comparison of homomorphic evaluation of several block cipher circuits

分组密码	实现方式	加密比特数	MGSW. Mult 次数	MGSW. Mult/bit
AES-128	逐字节切割	$8r$	912	$114/r$
	逐状态切割	$128r$	5120	$40/r$
PRINCE	逐状态切割	$64r$	1920	$30/r$
SIMON-64/128	逐状态切割	$64r$	1408	$22/r$
	半状态切割	$2r$	308	$154/r$

从表 2 可以看出，在设定相同参数 r 的情况下，逐状态切割效率优于其他实现方式，且同样用逐状态切割实现的情况下三种分组密码的实现效率从高到低依次为：SIMON-64/128＞PRINCE＞AES-128.

6.2　安全性比较

作为高级加密标准，AES 算法最突出的优点是安全性，目前针对 AES-128 算法的攻击中最优的时间复杂度是 $2^{125.56}$，且所需要的数据量为 2^{128}.[20] 而轻量级密码算法虽然实现效率高，但安全性比起 AES 算法较弱，如 PRINCE 算法，当攻击者得到用同一个未知密钥加密的 2^n 个明密文对时，求解该密钥的时间复杂度为 2^{127-n}，[21] 只要 n 足够大，时间复杂度可以降到很低. 总的来说，AES 的安全性明显优于 PRINCE 和 SIMON 算法，在具体应用中，可以按照需要选取合适的算法.

7　结　论

为了解决 FHE 方案严重的密文扩张给密文传输带来的压力，本文给出了基于 MGSW15 方案实现“密文压缩”的基本框架，得到的密文可以转化为任意 LWE 上 FHE 方案的密文，使得应用更加灵活，我们给出了分组密码 AES-128，PRINCE 和 SIMON-64/128 电路的同态运算过程，对于安全性需求较高的应用，可以选择 AES-128 算法，对于数据量比较大的情况，可以选择 SIMON-64/128 算法，以提高密文传输效率.

参考文献：

[1] Gentry C. Fully homomorphic encryption using ideal lattices [C]//Proceedings of the 41st annual ACM symposium on theory of computing-STOC 2009. [s. l.]: ACM Press, 2009.

[2] Yagisawa M. Fully homomorphic encryption without bootstrapping [J]. Acm Comput Theor, 2015, 6: 1.

[3] Brakerski Z. Fully homomorphic encryption without modulus switching from classical gapSVP [J]. Lect Notes Comp Sci, 2012, 7417: 868.

[4] Brakerski Z, Vaikuntanathan V. Efficient fully homomorphic encryption from (Standard) LWE [J]. SIAM J Comput, 2014, 43: 831.

[5] Brakerski Z, Vaikuntanathan V. Fully homomorphic encryption from ring-LWE and security for key dependent messages [C]// Cryptology Conference. Berlin, Heidelberg: Springer, 2011.

[6] Dijk M V, Gentry C, Halevi S, *et al*. Fully homomorphic encryption over the integers [M]. Berlin, Heidelberg: Springer, 2010.

[7] Gentry C, Sahai A, Waters B. Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based [M]. Berlin, Heidelberg: Springer, 2013.

[8] Hiromasa R, Abe M, Okamoto T. Packing messages and optimizing bootstrapping in GSW-FHE

[C]// IACR International Workshop on Public Key Cryptography. Berlin, Heidelberg: Springer, 2015.

[9] Naccache D, Tibouchi M. Public key compression and modulus switching for fully homomorphic encryption over the integers [C]//International Conference on Theory and Applications of Cryptographic Techniques. Berlin, Heidelberg: Springer-Verlag, 2012.

[10] Gentry C, Halevi S, Smart N P. Fully homomorphic encryption with polylog overhead [M]. Berlin, Heidelberg: Springer, 2012.

[11] Cheon J H, Coron J S, Kim J, *et al*. Batch fully homomorphic encryption over the integers [C]// International Conference on the Theory and Applications of Cryptographic Techniques. Berlin, Heidelberg: Springer, 2013.

[12] Naehrig M, Lauter K, Vaikuntanathan V. Can homomorphic encryption be practical? [C]// ACM Cloud Computing Security Workshop. Chicago, USA: DBLP, 2011.

[13] Gentry C, Halevi S, Smart N P. Homomorphic evaluation of the AES circuit [J]. Lect Notes Comp Sci, 2012, 7417: 850.

[14] Canteaut A, Carpov S, Fontaine C, *et al*. Stream ciphers: a practical solution for efficient homomorphic-ciphertext compression [M]. Berlin, Heidelberg: Springer, 2016.

[15] Borghoff J, Canteaut A, Kavun E B, *et al*. PRINCE: a low-latency block cipher for pervasive computing applications [C]// International Conference on the Theory and Application of Cryptology and Information Security. Berlin, Heidelberg: Springer-Verlag, 2012.

[16] Beaulieu R, Shors D, Smith J, *et al*. The SIMON and SPECK families of lightweight block ciphers [EB/OL]. (2013-04-04)[2018-08-22]. http://eprint.iacr.org/2013/404.pdf.

[17] Alperin-Sheriff J, Peikert C. Faster bootstrapping with polynomial error [C]// International Cryptology Conference. Berlin, Heidelberg: Springer-Verlag, 2014.

[18] Boyar J, Peralta R. A new combinational logic minimization technique with applications to cryptology [J]. Lect Notes Comput Sci, 2010, 6049: 178.

[19] Eisenbarth T, Gong Z , Tim Güneysu, *et al*. Compact implementation and performance evaluation of block ciphers in ATtiny devices [C]// Progress in Cryptology-AFRICACRYPT 2012. Berlin: Springer Berlin Heidelberg, 2012.

[20] Bogdanov A, Chang D, Ghosh M, *et al*. Bicliques with minimal data and time complexity for AES [M]// Information Security and Cryptology-ICISC 2014. [s. l.]: [s. n.], 2014.

[21] Kilian J, Rogaway P. How to protect DES against exhaustive key search [C]// International Cryptology Conference on Advances in Cryptology. Berlin, Heidelberg: Springer-Verlag, 1996.

引用本文格式:

中 文: 刘帅, 胡斌. 基于 MGSW15 方案的分组密码电路的同态运算 [J]. 四川大学学报: 自然科学版, 2019, 56: 661.

英 文: Liu S, Hu B. Membership-function-dependent control of fuzzy systems with time-varying delays [J]. J Sichuan Univ: Nat Sci Ed, 2019, 56: 661.