基于整数的轻量级分组密码电路的同态运算

毛和风 胡 斌
(信息工程大学 郑州 450001)

摘 要 基于 Gentry 等在 EUROCRYPT 2010 上提出的整数上的全同态加密 DGHV 方案,结合批处理技术,给出了轻量级分组密码 SIMON 电路的状态切割同态运算实现方法;提出了半字节切割概念,给出了 PRINCE 电路的半字节切割同态运算实现方法。最后将 PRINCE,SIMON-64/128,SIMON-128/256 和 AES-128 电路的同态运算进行对比,分析给出了不同分组密码电路和不同实现方法的同态计算次数。

关键词 全同态加密,SIMON 电路,PRINCE 电路,同态运算
中图法分类号 TN918.1 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.11.026

Homomorphic Evaluation of Lightweight Block Cipher over Integers

MAO He-feng HU Bin
(Information Engineering University,Zhengzhou 450001,China)

Abstract Based on the fully homomorphic encryption DGHV scheme proposed by Gentry et al. in EUROCRYPT 2010 and the technology of batch,this paper presented a homomorphic evaluation method of lightweight block cipher SIMON circuit by state-wise bitslicing,and proposed a representation called half-byte-wise bitslicing. On this basis,this paper provided the implementation method of half-byte-wise bitslicing homomorphic evaluation of PRINCE circuit. Lastly,this paper compared PRINCE,SIMON-64/128,SIMON-128/256 with AES-128 with respect to the homomorphic operations,and analyzed the counts of homomorphic evaluation of different block cipher circuits and different implementation methods.

Keywords Fully homomorphic encryption,SIMON circuit,PRINCE circuit,Homomorphic evaluation

1 引言

全同态加密是一种功能强大的加密技术,能够在加密数据上执行任意计算,同时将对应的计算映射到明文中。全同态加密技术能够全密态处理数据。采用该加密技术可以将数据以加密形式传送给任何不可信服务器进行“密文计算”来获取服务,保证了数据的安全。全同态加密具有广阔的应用前景,可应用于云安全、加密数据库、搜索引擎的加密询问等。

1978 年,Rivest 等[1]提出了隐私同态的概念,隐私同态可以确保被操作数据的私密性,在不知道明文的情况下,可以直接操作密文,对被操作后的密文进行解密,得到的结果与对明文进行同样操作后的结果等价。全同态加密思想自提出以来,便受到了国内外学者的广泛关注,但他们所提出的方案均不具备全同态特性,即不能对密文进行任意次数的操作和处理。直到 2009 年,Gentry 基于理想格提出了第一个全同态加密方案[2](FHE),其主要思路是:首先构造一个 somewhat 同态加密方案,其只能够进行有限次的同态计算;然后对其解密电路进行“压缩”,降低其电路深度;最后进行自举转化,获得一个自举型方案。在进行同态运算时,利用重加密(同态解密)来更新密文,降低密文中的噪声,从而获得全同态加密方案。

在 Gentry 研究的影响下,全同态加密成为了热门的研究领域,学者们相继提出了很多基于不同困难问题的全同态加密方案[3-5],同时对 Gentry 提出的铜态加密方案中的技术进行改进。

由于全同态加密方案能够在任何布尔电路或运算函数上对密文做任意的运算,且运算结果的解密等同于对明文做相同运算的结果,因此经过全同态方案加密后的数据允许不可信服务器在没有解密私钥的情况下做任何运算操作。这一良好特性使其在云计算、加密数据库以及密文检索等领域具有广阔的应用前景。但是,采用目前的全同态加密方案获得的密文膨胀过高,使得用户在向云端传输数据时的代价太大,因此实用性稍差。

为了解决该问题,文献[6]提出可以使用分组密码(如 AES)加密数据后再发送到云端。云端利用同态加密方案对这些经过分组密码加密后的数据进行同态加密,然后再利用分组密码的逆算法电路(如 AES^{-1})对同态加密后的密文进行同态运算,即可得到用户原始数据的同态加密结果,将其存储在云端,以便于加密数据的查询及其他计算。在这种模式

到稿日期:2017-10-16 返修日期:2018-01-06 本文受国家自然科学基金(61702548)资助。
毛和风(1993-),女,硕士生,主要研究方向为全同态密码,E-mail:maohf8844@126.com;**胡 斌**(1971-),男,教授,博士生导师,主要研究方向为密码学与信息安全,E-mail:hb2110@126.com(通信作者)。

下,用户在网络中只需传输分组密码加密的数据,即可得到云端同态加密后的结果,因此网络通信传输量被降低到数据大小,FHE 公钥和同态加密过的分组密码的密钥仅需要一次传输并被存储在云端(见图 1)。

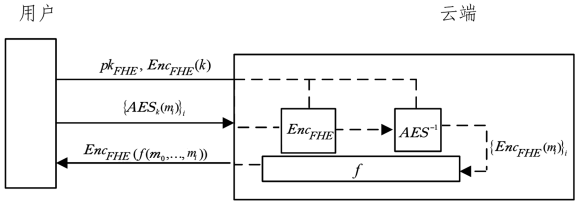

图 1 全同态加密系统与云端通信的示意图

Fig. 1 Schematic map of fully homomorphic cryptography system communicating with cloud

此方案中对分组密码算法电路进行同态运算的思想已经在 AES 上得到了实现。文献[7]和文献[8-9]分别利用基于格和基于整数的全同态加密方案实现了 AES 电路的同态运算,在实现过程中使用了字节切割技术[10-11]和批处理技术。

轻量级密码算法适用于资源受限的应用环境。随着密码应用领域的不断拓展,轻量级密码近年来受到了密码研究者的广泛关注。2013 年 7 月,美国国家安全局提出了一种轻量级分组密码 SIMON[12],它具有标准 Feistel 结构且设计小巧,易于硬件实现,另外,其圈函数只含有一个二进制 AND 运算。PRINCE[14]算法采用了 SPN 结构,具有良好的硬件结构,只需很小的存储空间。该算法具有 12 圈轮函数,其圈密钥是常数且采用 4×4 的 S 盒。

Coron 等[8]提出了基于整数的 AES 电路的同态运算,并给出了基于字节切割和状态切割的同态实现方法。本文基于 DGHV 方案,结合批处理技术,研究并给出了 SIMON 电路和 PRINCE 电路的同态运算实现方法。其中,主要给出了 SIMON 电路的状态切割同态运算实现方法,并提出了半字节切割概念;同时,还给出了 PRINCE 电路的半字节切割的同态运算实现方法。最后,将 AES-128,SIMON-64/128,SIMON-128/256 和 PRINCE 电路的同态运算进行对比,并给出了不同分组密码电路和不同实现方法的同态计算次数。

2 预备知识

Coron 等对 DGHV 方案[4]进行了同态批加密,把 ℓ 个明文打包加密为单个密文,得到整数上的批处理全同态加密方案[8](BDGHV)。

2.1 BDGHV 全同态加密方案

BDGHV 全同态加密方案是使用中国剩余定理对 Gentry 提出的整数上的全同态加密方案进行批同态加密处理而得到的。该方案的具体描述如下:

1)BDGHV. KeyGen(1^λ) 生成随机素数集合 $p_0,\cdots,p_{\ell-1}$,其中 p_i 的位长为 η,并将它们的乘积表示为 π。定义一个没有噪声的公钥元素 $x_0=q_0\cdot\pi$,其中 q_0 满足:$q_0\leftarrow\mathbb{Z}\cap[0,2^\gamma/\pi)$,不包含小于 2^{λ^2} 的素数因子。

在 $\mathbb{Z}\cap[0,q_0)$ 上均匀、独立地生成整数 x_i,x_i' 和 Π_i,当 $0\leqslant j<\ell$ 时,满足:

$1\leqslant i\leqslant\tau, x_i \bmod p_j=2r_{i,j}, r_{i,j}\leftarrow\mathbb{Z}\cap(-2^{\rho'-1},2^{\rho'-1})$

$0\leqslant i\leqslant\ell-1, x_i' \bmod p_j=2r'_{i,j}+\delta_{i,j}, r'_{i,j}\leftarrow\mathbb{Z}\cap(-2^{\rho},2^{\rho})$

$0\leqslant i\leqslant\ell-1, \Pi_i \bmod p_j=2\bar{\omega}_{i,j}+\delta_{i,j}\cdot 2^{\rho'+1}, \bar{\omega}_{i,j}\leftarrow\mathbb{Z}\cap(-2^{\rho},2^{\rho})$

令 $pk^*=\langle x_0,(x_i)_{1\leqslant i\leqslant\tau},(x_i')_{0\leqslant i\leqslant\ell-1},(\Pi_i)_{0\leqslant i\leqslant\ell-1}\rangle$,$sk^*=(p_j)_{0\leqslant j\leqslant\ell-1}$。设 $x_{p_j}\leftarrow\lceil 2^\kappa/p_j\rfloor$,$j=0,\cdots,\ell-1$。任选一组随机 Θ 比特向量 $s_j=(s_{j,0},\cdots,s_{j,\Theta-1})$,向量中每一个分量的汉明重量为 θ,$j=0,\cdots,\ell$。任选 Θ 比特整数 $u_i\in[0,2^{\kappa+1})$,$0\leqslant i<\Theta$,对于任意的 j,可以得到 $x_{p_j}=\sum_{i=1}^{\Theta-1}s_{j,i}\cdot u_i \bmod 2^{\kappa+1}$,同时有 $y_i=u_i/2^\kappa$,$y=(y_0,\cdots,y_{\Theta-1})$。因此,每个分量 y_i 都是小于 2 的正数,且其精度保留至二进制小数点后 κ 位。得到:

$$1/p_j=\sum_{i=0}^{\Theta-1}s_{j,i}\cdot y_i+\varepsilon_j \bmod 2$$

其中,$|\varepsilon_j|<2^{-\kappa}$。

输出私钥 $sk=(s_0,\cdots,s_{\ell-1})$,公钥 $pk=(pk^*,y_0,\cdots,y_{\Theta-1})$。

2)BDGHV. Encrypt(pk,$m\in\{0,1\}^\ell$) 选取任意整数向量 $b'=(b_i')_{0\leqslant i\leqslant\ell-1}\in(-2^{\alpha'},2^{\alpha'})^\ell$ 和 $b=(b_i)_{1\leqslant i\leqslant\tau}\in(-2^\alpha,2^\alpha)^\tau$,输出密文 $c=[\sum_{i=0}^{\ell-1}m_i\cdot x_i'+\sum_{i=0}^{\ell-1}b_i'\cdot\Pi_i+\sum_{i=1}^{\tau}b_i\cdot x_i]_{x_0}$。

3)BDGHV. Add(pk,c_1,c_2) 输出 $c_1+c_2 \bmod x_0$。

4)BDGHV. Mult(pk,c_1,c_2) 输出 $c_1\cdot c_2 \bmod x_0$。

5)BDGHV. Expand(pk,c) 对于每一个 $0\leqslant i\leqslant\Theta-1$,计算得到 $z_i=\lceil c\cdot y_i\rfloor \bmod 2$,且其精度保留至二进制小数点后 $n=\lceil\log_2(\theta+1)\rceil$ 位。定义向量 $z=(z_i)_{i=0,\ldots,\Theta-1}$,输出扩展密文 (c,z)。

6)BDGHV. Decrypt(sk,c,z) 输出明文 $m=(m_0,\cdots,m_{\ell-1})$,其中:

$$m_j\leftarrow[\lceil\sum_{i=0}^{\Theta-1}s_{j,i}\cdot z_i\rfloor]_2\oplus(c \bmod 2)$$

7)BDGHV. Recrypt(pk,c,z) 使用加密过的私钥对扩展密文 z 脱密,输出新鲜密文 c_{new}。

2.2 参数定义

给定安全参数 λ,并定义公钥的元素个数 τ、公钥位长 γ、私钥位长 η、噪声位长 ρ、加密过程中的次噪声 ρ'。对于一个实数 x,$\lceil x\rceil$,$\lfloor x\rfloor$ 和 $\lceil x\rfloor$ 分别表示对实数 x 向上、向下和就近取整。对于实数 z 和 p,用 $(z \bmod p)$ 或 $[z]_p$ 表示 z 除以 p 的余数,其中 $-p/2<[z]_p\leqslant p/2$。

方案中的各个参数必须满足以下约束条件:为了避免针对噪声的暴力攻击,$\rho\geqslant 2\lambda$;为了保证解密的正确性,$\eta\geqslant\alpha'+\rho'+1+\log_2(\ell)$;为了同态密文计算解密压缩电路,$\eta\geqslant\rho\cdot\Theta(\lambda\log^2\lambda)$;为了防止基于格的攻击,$\gamma=\omega(\eta^2\cdot\log\lambda)$;为了保证方案的语义安全,$\rho'\geqslant\rho+\lambda$ 和 $\alpha'\geqslant\alpha+\lambda$;为了应用哈希剩余引理,$\alpha\cdot\tau\geqslant\gamma+\lambda$ 和 $\tau\geqslant\ell\cdot(\rho'+2)+\lambda$。

为了满足上述约束条件,可以选取如下参数:$\rho=2\lambda$,$\eta=\tilde{\mathcal{O}}(\lambda^2)$,$\gamma=\tilde{\mathcal{O}}(\lambda^5)$,$\alpha=\tilde{\mathcal{O}}(\lambda^2)$,$\tau=\tilde{O}(\lambda^3)$,$\rho'=3\lambda$,$\alpha'=\tilde{\mathcal{O}}(\lambda^2)$,$\ell=\tilde{\mathcal{O}}(\lambda^2)$。

2.3　明文向量上的同态置换

由 BDGHV 方案可知，可以在打包密文上实现同态运算，即 Add 和 Mult 可以同时作用在 ℓ 个不同的输入比特上。可以利用同态置换技术来实现打包密文中不同明文槽间数据的移动与运算。同态置换可以在不“解包”的情况下，同态地批处理密文，同时不会产生任何额外的同态运算消耗。

对于$\{0,\cdots,\ell-1\}$上的任意置换 ζ，同态置换运算可表示为：

$$\ell\text{-Permute}(\zeta,(u_0,\cdots,u_{\ell-1}))=(u_{\zeta(0)},\cdots,u_{\zeta(\ell-1)})$$

其中，ζ 定义为同态作用在明文比特上的置换。在 KeyGen 阶段，对所有的 $i\in[0,\Theta-1]$，定义置换：

$\sigma_i^{\zeta}=\text{BDGHV.Encrypt}(s_{\zeta(0),i},\cdots,s_{\zeta(\ell-1),i})$ 可以使用 σ_i^{ζ} 在密文上进行置换运算。因此，通过在公钥中加入置换元素 σ_i^{ζ}，可以对密文进行同态置换运算，这相当于对相应的明文进行置换 ζ 运算。

2.4　轻量级分组密码 SIMON

2013 年 7 月，美国国家安全局公开了一组轻量级分组密码 SIMON[12]，它可以优化硬件性能。SIMON 是标准 Feistel 结构（见图 2），其分组长度为 $2n$（$n=16,24,32,48,64$）比特；密钥长度为 mn，根据不同的主密钥规模，m 取值 2，3 或 4，通常记作 SIMON-$2n/mn$。SIMON 所有不同参数的版本如表 1 所列。

表 1　SIMON 所有不同参数的版本

Table 1　Different parameter visions of SIMON

分组长度 $2n$	密钥长度 mn	主密钥规模 n	密钥参数 m	常数列	圈数 N_r
32	64	16	4	z_0	32
48	72	24	3	z_0	36
	96		4	z_1	36
64	96	32	3	z_2	42
	128		4	z_3	44
96	96	48	2	z_2	52
	144		3	z_3	54
128	128	64	2	z_2	68
	192		3	z_3	69
	256		4	z_4	72

以 SIMON-128/256 为例，其分组长度为 128 比特，密钥长度为 256 比特，圈数（N_r）为 32。

SIMON 算法的第 $i+1$ 轮迭代结构如图 2 所示。在第 $i+1$ 轮时，x_i 和 y_i 分别表示第 $i+1$ 轮输入的左半部分和右半部分；k_i 表示第 $i+1$ 轮密钥；x_{i+1} 和 y_{i+1} 分别表示第 $i+1$ 轮输出的左半部分和右半部分；F 表示轮函数，其中 $F(x)=((x\lll 8)\otimes(x\lll 1))\oplus(x\lll 2)$，$(x\lll j)$ 表示 x 循环左移 j 位，$\otimes$ 表示按比特 AND 运算，$\oplus$ 表示按比特异或运算。第 $i+1$ 轮结构可按如下表达式列出：

$$\begin{cases}x_{i+1}=F(x_i)\oplus y_i\oplus k_i\\ y_{i+1}=x_i\end{cases}$$

最后一轮的输出 x_{N_r} 和 y_{N_r} 即为密文。

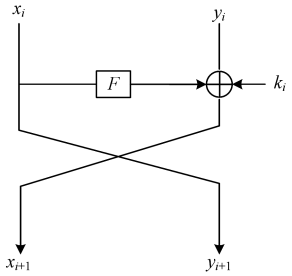

图 2　SIMON 圈函数

Fig. 2　SIMON round function

2.5　轻量级分组密码 PRINCE

PRINCE[13]算法是一个分组长度为 64 比特、密钥长度为 128 比特的轻量级分组密码算法。密钥可分为两部，分每部分长度均为 64 比特，可表示为：$k=k_0\parallel k_1$。其扩展密钥长度为 192 比特，由 k_0 生成 k_0'，表示为：

$$\begin{aligned}k&=k_0\parallel k_1\rightarrow(k_0\parallel k_0'\parallel k_1)\\&=(k_0\parallel(k_0\ggg 1)\oplus(k_0\gg 63)\parallel k_1)\end{aligned}$$

PRINCE 采用 SPN 结构，算法前后各进行一次白化密钥 k_0 和 k_0'，而 k_1 用于 12 圈轮函数，称之为 PRINCEcore。PRINCEcore 的结构图和 PRINCE 算法的加密过程分别如图 3 和图 4 所示。

图 3　PRINCEcore 的结构图

Fig. 3　Structure of PRINCEcore

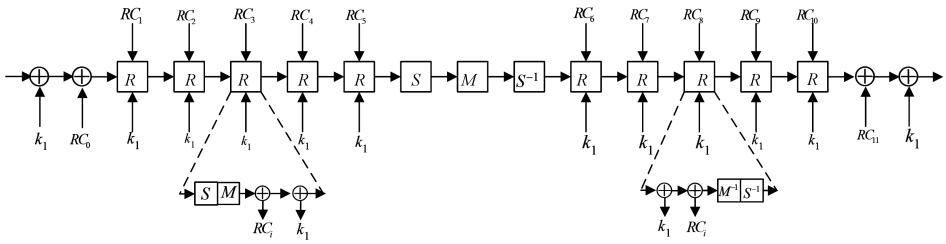

图 4　PRINCE 算法的加密过程

Fig. 4　Encryption process of PRINCE algorithm

PRINCE 轮函数 R 共包括 4 个操作：密钥加、常数加、矩阵乘及 S 盒。

1）密钥加：密钥 k_1 与状态对应 64 比特异或。

2）常数加：PRINCE 算法为每一圈定义了不同的圈常数 RC_i，在此阶段表示为常数 RC_i 与状态对应 64 比特异或。圈常数具有一个明显的特征：对于所有的 $0\leqslant i\leqslant 11$，$RC_i\oplus RC_{11-i}=\alpha$，$\alpha$ 是一个常数，且其值为 $\alpha=$ c0ac29b7c97c50dd。RC_i 的值如表 2 所列。

表 2 RC_i 的值

Table 2 Value of RC_i

RC_i	值
RC_0	0000000000000000
RC_1	13198a2e03707344
RC_2	a4093822299f31d0
RC_3	082efa98ec4e6c89
RC_4	452821e638d01377
RC_5	be5466cf34c90c6c
RC_6	7ef84f78fd955cb1
RC_7	85840851f1ac43aa
RC_8	c882d32f25323c54
RC_9	64a51195e0e3610d
RC_{10}	d3b5a399ca0c2399
RC_{11}	c0ac29b7c97c50dd

3)矩阵乘:在该阶段,64 比特状态与一个 64×64 的矩阵 M'相乘。为了保证信息扩散的完全性,将矩阵 M'与行移位运算(SR)结合起来,这里的 SR 与 AES 中的行移位运算相同,可以把二进制转换成十六进制,然后对 16 个分块进行位移。因此,此阶段的运算表示为 $M=SR\circ M'$。该阶段也可称作 P 置换。

定义:

$$\hat{M}^{(0)}=\begin{pmatrix} M_0 & M_1 & M_2 & M_3 \\ M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \end{pmatrix}$$

$$\hat{M}^{(1)}=\begin{pmatrix} M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \\ M_0 & M_1 & M_2 & M_3 \end{pmatrix}$$

其中:

$$M_0=\begin{pmatrix} 0&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix}, M_1=\begin{pmatrix} 1&0&0&0 \\ 0&0&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix}$$

$$M_2=\begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \end{pmatrix}, M_3=\begin{pmatrix} 0&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&0 \end{pmatrix}$$

构造矩阵 M'为一个 64×64 的对角型的对合矩阵,$M'=\mathrm{diag}(\hat{M}^0,\hat{M}^1,\hat{M}^1,\hat{M}^0)$。

4)S 盒:S 盒是 PRINCE 算法中唯一的非线性运算,PRINCE 采用 4×4 S 盒,且 S 盒为十六进制。S 盒的作用如表 3 所列。

表 3 S 盒

Table 3 S-box

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

在同态运算中,有两个主要因素会增加连续乘法的次数:1)S 盒的大小和复杂性,较高的非线度通常需要深度较高的电路,其中电路深度为连续二进制乘法的次数;2)轮函数的圈数。PRINCE 算法采用 4×4 的 S 盒,且只有 12 圈轮函数,这使得可以利用该算法电路进行同态运算。

3 轻量级分组密码电路的同态运算实现

3.1 SIMON 电路的状态切割同态运算实现

本节基于 DGHV 方案的批同态方案(即 BDGHV 方案),给出 SIMON 电路的同态运算实现。由于批同态运算利用了 SIMD 技术的思想[14],因此为了提高实现效率,通常可采用并行抽取特殊的比特来进行同态运算,即可通过结合比特切割技术与 SIMD 技术来提高实现效率。下面给出按状态的比特进行抽取的实现方法,即 SIMON 电路的状态切割同态运算实现方法。

3.1.1 状态切割表示

首先使用 BDGHV 方案把 ℓ 个明文比特嵌入进每一个密文中,用 $k=\ell$ 表示 SIMON 电路的并行同态运算的 SIMON 状态数量,即 k 个 SIMON 状态同时执行同态运算。

通过状态切割,可以将 SIMON 电路的输入状态看作由 $2n$ 个密文组成,即 SIMON 电路的输入状态可以写为 $c_1,\cdots,c_n,c_{n+1},\cdots,c_{2n}$,其对应的明文为 $m_1,\cdots,m_n,m_{n+1},\cdots,m_{2n}$。用 $m_i[k]$表示第 k 个 SIMON 输入状态的第 i 比特,即运用批处理技术[7],将 SIMON 输入状态的第 i 个明文编码到相应的第 i 个槽中。这样,将 k 个 SIMON 输入状态的 $2n\cdot k$ 比特数据表示成 $2n$ 个不同的密文,也即把 k 个不同的 SIMON 状态放入 $2n$ 个密文中。批处理过程如下:

首先,把 k 个 SIMON 状态写成 $2n$ 比特的向量形式:

第 1 个 SIMON 状态:$(s_{1,1},s_{1,2},\cdots,s_{1,n},s_{1,n+1},\cdots,s_{1,2n})$

第 2 个 SIMON 状态:$(s_{2,1},s_{2,2},\cdots,s_{2,n},s_{2,n+1},\cdots,s_{2,2n})$

第 3 个 SIMON 状态:$(s_{3,1},s_{3,2},\cdots,s_{3,n},s_{3,n+1},\cdots,s_{3,2n})$

…

第 k 个 SIMON 状态:$(s_{k,1},s_{k,2},\cdots,s_{k,n},s_{k,n+1},\cdots,s_{k,2n})$

则 SIMON 的状态切割所对应的明文可表示为:

$m_1=(s_{1,1},s_{2,1},s_{3,1},\cdots,s_{k,1})$

$m_2=(s_{1,2},s_{2,2},s_{3,2},\cdots,s_{k,2})$

…

$m_n=(s_{1,n},s_{2,n},s_{3,n},\cdots,s_{k,n})$

$m_{n+1}=(s_{1,n+1},s_{2,n+1},s_{3,n+1},\cdots,s_{k,n+1})$

…

$m_{2n}=(s_{1,2n},s_{2,2n},s_{3,2n},\cdots,s_{k,2n})$

由 BDGHV 方案加密可得密文 $c_1,\cdots,c_n,c_{n+1},\cdots,c_{2n}$,即每个密文中都包含且只包含每个 SIMON 状态中的一个比特。

因此,SIMON 输入状态的左半部分 $x=(x_1,\cdots,x_n)\in\mathbb{F}_2^n$加密为 $c_1,\cdots,c_n$,右半部分 $y=(y_1,\cdots,y_n)\in\mathbb{F}_2^n$加密为 $c_{n+1},\cdots,c_{2n}$,其中 $1\leqslant j\leqslant n$。

3.1.2 状态切割同态运算

下面使用状态切割的方式对 SIMON 电路进行同态运算。

首先,对左半部分密文进行函数 F 运算。函数 F 中的循环移位不再作用在明文上,而是作用在同态 SIMON 状态的密文下标上,即对同态 SIMON 状态的密文重新标记下标。循环移位 a 位可以通过改变下标实现,写为 $c_{(i+a)\bmod n}$。通过

改变密文下标，可以实现 n 比特明文循环移位 a 位。因此，该运算不消耗任何同态运算。

循环移位后，两个状态进行 AND 运算，即一个状态每一比特与另一状态相对应的比特进行 AND 运算。在同态运算中，AND 运算是通过同态乘法运算实现的，即两个状态密文通过 BDGHV. Mult 进行同态乘法运算，这相当于其对应的明文比特进行 AND 运算。值得注意的是，这里是 k 个 SIMON 电路同时进行同态乘法运算。

对于两个状态的 XOR 运算，即两个状态相对应的比特进行 XOR 运算，在同态运算中，可以通过两个同态密文的同态加法运算来实现，即两个状态密文的相应比特通过 BDGHV. Add 进行同态加法运算，这相当于其对应的明文比特进行 XOR 运算，且 k 个 SIMON 电路同时进行同态加法运算。

因此，SIMON 状态左半部分密文 $c_1,\cdots,c_n$ 进行函数 F 运算可以表示为：

$$F(c_j)=(c_{(j+8)\bmod n}\cdot c_{(j+1)\bmod n})+c_{(j+2)\bmod n}$$

因此，函数 F 运算包含 n 个 BDGHV. Mult 运算和 n 个 BDGHV. Add 运算。

圈密钥 k_i 的密钥比特可以表示为 k_{ij}，其加密形式记为 e_{ij}。状态的右半部分与圈密钥作 XOR 运算可以用 n 个同态加法运算表示，即圈密钥比特的加密与状态密文相对应的比特通过 BDGHV. Add 进行同态加法运算，记为 $c_{n+j}\leftarrow c_{n+j}+e_{ij}$，这相当于其对应的明文比特进行 XOR 运算，且 k 个 SIMON 分组同时进行同态运算。该运算共包含 n 个 BDGHV. Add 运算。

因此，由状态 $c_1,\cdots,c_{2n}$ 到状态 $c_1',\cdots,c_{2n}'$，可以做如下运算：

$$c'_{n+j}\leftarrow c_j$$

$$c_j'=(c_{(j+8)\bmod n}\cdot c_{(j+1)\bmod n})+c_{(j+2)\bmod n}+e_{ij}+c_{n+j}$$

由上式可知，对 SIMON 电路每进行一圈同态运算，就进行一次同态乘法运算，因此同态加密方案的电路深度不少于 SIMON 电路的运算圈数，即运算 SIMON-128/256 的同态电路深度至少为 72。

由于每次同态乘法运算后都会产生噪声，为使同态运算正常运行，需要在每次同态乘法后将状态恢复到一个较小的噪声，即需要进行 BDGHV. Recrypt 运算。通过计算可得进行一次加密后的噪声为：

$$c \bmod p_j=\sum_{i=0}^{\ell-1}m_i\cdot(2r'_{i,j}+\delta_{i,j})+\sum_{i=1}^{\tau}b_i\cdot 2r_{i,j}+\sum_{i=0}^{\ell-1}b_i'\cdot(2\omega_{i,j}+\delta_{i,j}\cdot 2^{\rho'+1})$$

计算可得噪声的量级约为 $\widetilde{O}(\lambda)$。因为同态密文经过同态乘法运算后其噪声的量级是以指数倍数的形式增长的，所以两次同态乘法运算后密文噪声的量级为 $\widetilde{O}(\lambda^3)$，然而 p_i 的位长为 η，其量级为 $\widetilde{O}(\lambda^2)$，这小于密文两次同态乘法运算后的密文噪声的量级，故同态密文无法进行连续的同态乘法运算。因此，每进行一次 BDGHV. Mult 运算，必须进行一次 BDGHV. Recrypt 运算。

同态计算的次数为：通过以上计算，一圈 SIMON 同态运算过程共包含 $3n$ 个 BDGHV. Add 运算、n 个 BDGHV. Mult 运算和 n 个 BDGHV. Recrypt 运算。以 SIMON-128/256 为例，同态运算过程共包含 13824 个 BDGHV. Add 运算、4608 个 BDGHV. Mult 运算和 4608 个 BDGHV. Recrypt 运算。

3.2　PRINCE 电路的半字节切割同态运算实现

状态：设 $S=s_0s_1\cdots s_{15}$ 表示 64 比特的 PRINCE 的中间状态，可以按序排列成 4×4 的状态矩阵，每一个元素包含 4 比特，即可表示为 $s_i=s_{i,0}s_{i,1}s_{i,2}s_{i,3}$，如图 5 所示。

$$\begin{pmatrix}s_0 & s_4 & s_8 & s_{12}\\ s_1 & s_5 & s_9 & s_{13}\\ s_2 & s_6 & s_{10} & s_{14}\\ s_3 & s_7 & s_{11} & s_{15}\end{pmatrix}_{4\times4}$$

图 5　PRINCE 的状态矩阵

Fig. 5　State matrix of PRINCE

使用 BDGHV 方案把 ℓ 个明文比特嵌入进每一个密文中，用 $k=\lfloor\ell/16\rfloor$ 表示并行处理的 PRINCE 状态数量，即 k 个 PRINCE 状态可以同时执行同态运算。由于 PRINCE 分组密码采用的是 4 比特 S 盒，参考文献[8]中的字节切割同态运算实现的方法，这里采用半字节切割实现方法。按半字节切割同态运算实现方法，PRINCE 电路可由 4 个密文 c_0,c_1,c_2,c_3 组成，其对应的明文是 m_0,m_1,m_2,m_3，用 $m_i[k\cdot16+j]$ 表示第 k 个 PRINCE 状态的第 j 个元素的第 i 比特，且每个密文包含每个 PRINCE 状态的 16 比特。半字节切割方式如下：

将 PRINCE 的状态矩阵用比特的形式表示：

第 1 个 PRINCE 状态：$\begin{pmatrix}s^1_{0,1} & s^1_{0,2} & s^1_{0,3} & s^1_{0,4}\\ s^1_{1,1} & s^1_{1,2} & s^1_{1,3} & s^1_{1,4}\\ \vdots & \vdots & \vdots & \vdots\\ s^1_{15,1} & s^1_{15,2} & s^1_{15,3} & s^1_{15,4}\end{pmatrix}_{16\times4}$

第 2 个 PRINCE 状态：$\begin{pmatrix}s^2_{0,1} & s^2_{0,2} & s^2_{0,3} & s^2_{0,4}\\ s^2_{1,1} & s^2_{1,2} & s^2_{1,3} & s^2_{1,4}\\ \vdots & \vdots & \vdots & \vdots\\ s^2_{15,1} & s^2_{15,2} & s^2_{15,3} & s^2_{15,4}\end{pmatrix}_{16\times4}$

…

第 k 个 PRINCE 状态：$\begin{pmatrix}s^k_{0,1} & s^k_{0,2} & s^k_{0,3} & s^k_{0,4}\\ s^k_{1,1} & s^k_{1,2} & s^k_{1,3} & s^k_{1,4}\\ \vdots & \vdots & \vdots & \vdots\\ s^k_{15,1} & s^k_{15,2} & s^k_{15,3} & s^k_{15,4}\end{pmatrix}_{16\times4}$

则 PRINCE 的半字节切割所对应的明文可表示为以下形式，其中每个矩阵包含 16 个元素。

$$m_0=\begin{pmatrix}s^1_{0,1}\cdots s^k_{0,1} & s^1_{4,1}\cdots s^k_{4,1} & s^1_{8,1}\cdots s^k_{8,1} & s^1_{12,1}\cdots s^k_{12,1}\\ s^1_{1,1}\cdots s^k_{1,1} & s^1_{5,1}\cdots s^k_{5,1} & s^1_{9,1}\cdots s^k_{9,1} & s^1_{13,1}\cdots s^k_{13,1}\\ s^1_{2,1}\cdots s^k_{2,1} & s^1_{6,1}\cdots s^k_{6,1} & s^1_{10,1}\cdots s^k_{10,1} & s^1_{14,1}\cdots s^k_{14,1}\\ s^1_{3,1}\cdots s^k_{3,1} & s^1_{7,1}\cdots s^k_{7,1} & s^1_{11,1}\cdots s^k_{11,1} & s^1_{15,1}\cdots s^k_{15,1}\end{pmatrix}$$

$$m_1=\begin{pmatrix}s^1_{0,2}s^2_{0,2}\cdots s^k_{0,2} & \cdots & s^1_{12,2}s^2_{12,2}\cdots s^k_{12,2}\\ \vdots & \ddots & \vdots\\ s^1_{3,2}s^2_{3,2}\cdots s^k_{3,2} & \cdots & s^1_{15,2}s^2_{15,2}\cdots s^k_{15,2}\end{pmatrix}$$

$$m_2=\begin{pmatrix}s^1_{0,3}s^2_{0,3}\cdots s^k_{0,3} & \cdots & s^1_{12,3}s^2_{12,3}\cdots s^k_{12,3}\\ \vdots & \ddots & \vdots\\ s^1_{3,3}s^2_{3,3}\cdots s^k_{3,3} & \cdots & s^1_{15,3}s^2_{15,3}\cdots s^k_{15,3}\end{pmatrix}$$

$$m_3=\begin{pmatrix} s^1_{0,4}\,s^2_{0,4}\cdots s^k_{0,4} & \cdots & s^1_{12,4}\,s^2_{12,4}\cdots s^k_{12,4} \\ \vdots & \ddots & \vdots \\ s^1_{3,4}\,s^2_{3,4}\cdots s^k_{3,4} & \cdots & s^1_{15,4}\,s^2_{15,4}\cdots s^k_{15,4} \end{pmatrix}$$

由 BDGHV 方案加密可得密文 c_0,c_1,c_2,c_3,即每个密文都包含每个 PRINCE 状态的 16 比特。

密钥加:每一圈轮函数中,密钥 k_1 与状态对应 64 比特异或,在此阶段,构建圈密钥结构使其与同态 PRINCE 状态的密文结构一致(即 4 个密文的状态)。在同态运算中,把圈密钥与密文通过 BDGHV. Add 进行运算,相当于密钥与明文进行 XOR 运算,且 k 个 PRINCE 状态同时进行同态运算。此阶段共包含 4 个 BDGHV. Add 运算。

常数加:每一圈轮函数中,圈常数 RC_i 与状态对应 64 比特异或,在此阶段,与密钥加相同,构建圈常数结构使其与同态 PRINCE 状态的密文结构一致(即 4 个密文的状态)。在同态运算中,将圈常数与密文通过 BDGHV. Add 进行运算,且 k 个 PRINCE 状态同时进行同态运算。此阶段共包含 4 个 BDGHV. Add 运算。

矩阵乘:在该阶段,状态密文先进行 SR,然后再与一个 64×64 的矩阵 M' 相乘。SR 与 AES 中的行移位运算相同。

行移位运算是在 PRINCE 状态矩阵上进行置换 ζ,如表 4 所列。

表 4 置换 ζ

Table 4 Permutation ζ

s_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\zeta(s_i)$	0	13	10	7	4	1	14	11	8	5	2	15	12	9	6	3

使用半字节切割的方式对 PRINCE 进行同态运算时需要将矩阵中每个半字节的 4 个比特分别放进不同的密文中,从而需要定义一个新的置换使其作用于密文。由于要实现 k 个 PRINCE 状态同时进行同态运算,因此定义置换 ζ' 为:

$$\zeta'(I\times k+K)=\zeta(I)\times k+K,0\leqslant K\leqslant k-1,0\leqslant I\leqslant 15$$

在 SR 阶段,用置换 ζ' 对 PRINCE 状态的密文进行同态置换,这相当于置换 ζ 作用在明文上。由 2.3 节可知,这可以通过在公钥中加入置换元素 ζ 实现,且置换运算没有任何同态运算消耗。

SR 运算后,状态密文与矩阵 M' 相乘,这就相当于两个同态密文进行乘法运算,即通过 BDGHV. Mult 进行同态乘法运算。每一次同态乘法运算后,密文中的噪声会以指数倍数的形式增长,即乘法运算后需要 BDGHV. Recrypt 运算将密文恢复到低噪音状态。由 3.1.2 节的计算可知,每进行一次 BDGHV. Mult 运算,必须进行一次 BDGHV. Recrypt 运算,因此此阶段共包含 64 个 BDGHV. Mult 运算和 64 个 BDGHV. Recrypt 运算。

S 盒:该算法中的唯一一个非线性运算为 S 盒,可以用 XOR 运算和 AND 运算来实现 S 盒的运算,即通过 BDGHV. Add 运算和 BDGHV. Mult 运算来实现 S 盒的同态运算。

PRINCE 算法中的 S 盒为十六进制,可以用 4 个二进制量表示,那么在同态运算中,S 盒运算可以写成:

$$S(s_{i,0},s_{i,1},s_{i,2},s_{i,3})=(s'_{i,0},s'_{i,1},s'_{i,2},s'_{1,3})$$

4 个比特通过 XOR 运算和 AND 运算可以得到 S 盒的运算结果,运算过程如图 6 所示。

$s'_{i,0}$	$s_{i,0}\oplus s_{i,2}\oplus s_{i,0}s_{i,1}\oplus s_{i,1}s_{i,2}\oplus s_{i,0}s_{i,1}s_{i,3}\oplus s_{i,0}s_{i,2}s_{i,3}\oplus s_{i,1}s_{i,2}s_{i,3}\oplus 1$
$s'_{i,1}$	$s_{i,0}\oplus s_{i,3}\oplus s_{i,0}s_{i,2}\oplus s_{i,0}s_{i,3}\oplus s_{i,2}s_{i,3}\oplus s_{i,0}s_{i,1}s_{i,2}\oplus s_{i,0}s_{i,2}s_{i,3}$
$s'_{i,2}$	$s_{i,0}s_{i,2}\oplus s_{i,1}s_{i,2}\oplus s_{i,1}s_{i,3}\oplus s_{i,0}s_{i,1}s_{i,2}\oplus s_{i,1}s_{i,2}s_{i,3}\oplus 1$
$s'_{i,3}$	$s_{i,0}\oplus s_{i,1}\oplus s_{i,0}s_{i,1}\oplus s_{i,0}s_{i,3}\oplus s_{i,1}s_{i,2}\oplus s_{i,2}s_{i,3}\oplus s_{i,1}s_{i,2}s_{i,3}\oplus 1$

图 6 S 盒的运算结果

Fig. 6 Computation results of S-box

由图 6 可知,S 盒需要 28 个 BDGHV. Mult 运算,可以采取重用中间项的方法对其进行进一步优化,以有效减少乘法的运算次数。观察表中运算,$s_{i,0}s_{i,1}$,$s_{i,0}s_{i,2}$,$s_{i,0}s_{i,3}$,$s_{i,1}s_{i,2}$,$s_{i,1}s_{i,3}$,$s_{i,2}s_{i,3}$ 的值可以计算并存储,S 盒运算时可以直接读取而不用重复计算,还有 4 个三项的乘法可以存储并反复利用,即 $s_{i,0}s_{i,1}s_{i,3}$,$s_{i,0}s_{i,1}s_{i,2}$,$s_{i,0}s_{i,2}s_{i,3}$,$s_{i,1}s_{i,2}s_{i,3}$。为了进一步提高效率,在计算三项乘法时,可以使用存储的两项乘法的值。这样,电路乘法的深度只有 2,即一层为了计算两项乘法,如 $s_{i,0}s_{i,1}$;另一层为了计算三项乘法,如 $s_{i,0}s_{i,1}s_{i,3}$。因此,在该阶段需要 10 个 BDGHV. Mult 运算、25 个 BDGHV. Add 运算和 10 个 BDGHV. Recrypt 运算。

通过以上计算,一圈 PRINCE 同态运算过程共包括 33 个 BDGHV. Add 运算、74 个 BDGHV. Mult 运算和 74 个 BDGHV. Recrypt 运算。因此,完整的 PRINCE 同态运算过程共包括 396 个 BDGHV. Add 运算、824 个 BDGHV. Mult 运算和 824 个 BDGHV. Recrypt 运算。

3.3 PRINCE 电路的状态切割同态运算实现

使用 BDGHV 方案把 ℓ 个明文比特嵌入进每一个密文中,用 $k=\ell$ 表示 PRINCE 电路的并行同态运算的 PRINCE 状态数量,即 k 个 SIMON 状态同时执行同态运算。

通过状态切割,可以将 PRINCE 电路的输入状态看作是由 64 个密文组成,即 PRINCE 电路的输入状态可以写为 $c_0,\cdots,c_{63}$,其对应的明文为 $m_0,\cdots,m_{63}$。用 $m_{i+j\cdot 4}[k]$ 表示第 k 个 PRINCE 输入状态的第 j 个元素的第 i 比特,即将 k 个 PRINCE 输入状态的 $64\cdot k$ 比特数据表示成 64 个不同的密文,也即把 k 个不同的 PRINCE 状态放入 64 个密文中。其批处理过程与 3.1 节中 SIMON 电路的批处理过程相同。

密钥加:每一圈轮函数中,密钥 k_1 与状态对应 64 比特异或,在同态运算中,通过 BDGHV. Add 对圈密钥与密文进行运算,此阶段共包含 4×16=64 个 BDGHV. Add 运算。

常数加:每一圈轮函数中,圈常数 RC_i 与状态对应 64 比特异或,在同态运算中,把圈常数与密文通过 BDGHV. Add 进行运算,此阶段共包含 4×16=64 个 BDGHV. Add 运算。

矩阵乘:在该阶段,状态密文先进行 SR,然后再与一个 64×64 的矩阵 M' 相乘。循环移位不再作用在明文上,而是作用在同态 PRINCE 状态密文的下标上,即对同态 PRINCE 状态的密文重新标记下标,以实现明文的置换。因此,这个运算不消耗任何同态运算。

SR 运算后,状态密文再与矩阵 M' 相乘,即通过 BDGHV. Mult 进行同态乘法运算。因此,此阶段共包含 64×16=1024

个 BDGHV. Mult 运算、64×16＝1024 个 BDGHV. Recrypt 运算。

S 盒：与 3.2 节相同，可以用 XOR 运算和 AND 运算来实现 S 盒的运算，即通过 BDGHV. Add 运算和 BDGHV. Mult 运算来实现 S 盒的同态运算。但本节的同态运算实现技术与 3.2 节不同，我们需要对 PRINCE 状态的每个密文进行 S 盒运算。因此，在这个阶段需要 10×16＝160 个 BDGHV. Mult 运算、25×16＝400 个 BDGHV. Add 运算和 10×16＝160 个 BDGHV. Recrypt 运算。

通过以上计算可知，完整的 PRINCE 同态运算过程共包括 6336 个 BDGHV. Add 运算、13184 个 BDGHV. Mult 运算和 13184 个 BDGHV. Recrypt 运算。

3.4　不同算法电路同态运算的实现代价

表 5 比较了 PRINCE，SIMON-64/128，SIMON-128/256 和 AES-128[8] 的不同分组密码算法电路同态运算时的计算次数，4 种方案均基于整数上的批处理全同态加密方案，且 AES 电路分别使用状态切割技术和字节切割技术进行同态运算。

表 5　PRINCE，SIMON-64/128，SIMON-128/256 和 AES-128 的同态运算次数

Table 5　Counts of homomorphic evaluation of PRINCE, SIMON-64/128, SIMON-128/256 and AES-128

方案	实现方法	加密密文比特	BDGHV. Add	BDGHV. Mult	BDGHV. Recrypt
PRINCE	半字节切割	64	396	824	824
PRINCE	状态切割	64	6336	13184	13184
SIMON-64/128	状态切割	64	4224	1408	1408
SIMON-128/256	状态切割	128	9216	4608	4608
AES-128	状态切割	128	14688	5120	2720
AES-128	字节切割	128	1260	320	386

结束语　本文基于 DGHV 方案，结合批处理技术，讨论了如何对 SIMON 电路和 PRINCE 电路实现同态运算，分析了其实现所需的同态运算次数；具体给出了 SIMON 电路的状态切割同态运算实现方法；提出了半字节切割技术，并基于此给出了 PRINCE 电路的半字节切割同态运算实现方法。相对于 Coron 等[8] 提出的基于整数的 AES 电路同态运算实现，基于轻量级分组密码电路的同态运算实现更简单，同态运算的次数更少。但是由于 BDGHV 方案自身的原因，重加密次数较多。通过 SIMON-128/256，SIMON-64/128，PRINCE 和 AES-128 的对比可以得出，SIMON 所需的电路深度较浅，同态运算量较小；PRINCE 结构简单，采用半字节切割实现时的同态运算量最小，状态切割同态运算方式可以同时运算的密文量大。按字节或半字节切割实现以及按状态切割同态运算实现是利用 SIMD 技术的两种不同实现方法，可根据应用中的实际需要选取，它们均可以提高同态运算的实现效率，且在云计算环境下有实际应用价值。是否有更好的基础方案，如何在同态运算实现中采取更好的置换实现方法，以及如何减少重加密的次数以更好地提升同态运算的实现效率，是值得进一步研究的问题。

参 考 文 献

[1] RIVEST R L, ADLEMAN L, DERTOUZOS M L. On data banks and privacy homomorphisms [J]. Foundations of Secure Computation, 1978, 4(11): 169-180.

[2] GENTRY C. Fully homomorphic encryption using ideal lattices [C]// Proc. of the 41st ACM Symposium on Theory of Computing. New York: ACM Press, 2009: 169-178.

[3] BRAKERSKI Z, GENTRY C, VAIKUNTANATHAN V. (Leveled) fully homomorphic encryption without bootstrapping [J]. ACM Transactions on Computation Theory(TOCT), 2014, 6(3): 13.

[4] VAN DIJK M, GENTRY C, HALEVI S, et al. Fully homomorphic encryption over the integers [M]// Advances in Cryptology-EUROCRYPT 2010. Berlin: Springer, 2010: 24-43.

[5] GENTRY C, SAHAI A, WATERS B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based [M]// Advances in Cryptology—CRYPTO 2013. Berlin: Springer, 2013: 75-92.

[6] NAEHRIG M, LAUTER K, VAIKUNTANATHAN V. Can homomorphic encryption be practical? [C]// Proc. of the 3rd ACM Workshop on Cloud Computing Security Workshop. New York: ACM Press, 2011: 113-124.

[7] GENTRY C, HALEVI S, SMART N. Homomorphic evaluation of the AES circuit [M]// Advances in Cryptology-CRYPTO 2012. Berlin: Springer, 2012: 850-867.

[8] CORON J S, LEPOINT T, TIBOUCHI M, et al. Batch fully homomorphic encryption over the integers [C]// Annual International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2013: 315-335.

[9] CORON J S, LEPOINT T, TIBOUCHI M. Scale-invariant fully homomorphic encryption over the integers [J]. IIar Journal, 2014, 50(4): 361-372.

[10] BIHAM E. A fast new DES implementation in software [C]// International Conference Fast Software Encryption. 1997: 260-271.

[11] KASPER E, SCHWABE P. Faster and timing－attack resistant AES-GCM[C]// Cryptographic Hardware and Embedded Systems-CHES. 2009: 1-17.

[12] BEAULIEU R, SHORS D, SMITH J, et al. The SIMON and SPECK families of lightweight block ciphers [EB/OL]. IACR Cryptology ePrint Archive. http://eprint.iacr.org/2013/404.pdf.

[13] SMART N P, VERCAUTEREN F. Fully homomorphic SIMD operations [J]. Designs, Codes and Cryptography, 2014, 71(1): 1-25.

[14] BORGHOFF J, CANTEAUT A, GüNEYSU T, 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: Springer, 2012: 208-225.