分组密码算法 FBC
设计与评估报告

(版本号：1.1)

冯秀涛、曾祥勇、张凡、曾光、唐灯、甘国华

2019.10.18

原创性声明

分组密码 FBC 算法族由本团队人员原创性设计，特此声明！

算法设计团队：冯秀涛、曾祥勇、张凡

曾光、唐灯、甘国华

版本修订记录

版本号	修订内容	修订日期
1.0	算法初稿	2019.03.25
1.1	调整 L 变换中的 s 和 t 参数取值	2019.10.18

目 录

1 概述

本报告主要描述了分组密码算法 FBC，并给出了其设计准则、部件特性、整体安全性评估和软硬件实现性能估算。

FBC 属于**轻量级分组密码算法**，包含 FBC128-128，FBC128-256 和 FBC256-256 三个版本，分别支持 128 比特和 256 比特两种明文分组以及 128 比特和 256 比特两种密钥长度。

FBC 算法采用 4 路两重 Feistel 结构设计，在结构上通过增加两个异或操作的微小代价较大提高整体结构的扩散特性，非线性函数 F 采用 bit-slice 技术，其中 S 盒基于 NFSR 构造，其各项密码学性质达到最优，同时硬件实现代价达到最小，为最轻的 S 盒之一，线性变换 L 仅由循环移位和异或构成，具有较好的密码学特性的同时兼顾好的软硬件实现效能。FBC 算法不仅具有结构简洁，轻量化，安全性高等特性，还具有灵活高效的软硬件实现方式，可以满足不同平台的应用需求。

本报告余下章节安排如下：

第 2 章描述了 FBC 算法规范；

第 3 章给出了算法设计准则；

第 4 章对算法部件特性给出了安全性分析；

第 5 章对算法整体安全性给出了评估；

第 6 章对算法软硬件性能作了简单估算。

2 算法规范

2.1 算法简介

本章描述了一种分组加密算法 FBC，其以英文 **F**eistel-based **B**lock **C**ipher 的首字母命名。该算法基于两重 Feistel 结构设计，支持 128 比特和 256 比特明文分组，以及 128 比特和 256 比特的主密钥，主要包含三个版本：FBC128-128, FBC128-256, FBC256-256。

2.2 算法主要参数

算法不同版本以格式 FBC*n*-*m* 统一进行命名，其中 n 为明文分组比特长度，m 为主密钥比特长度。不同版本的状态均由 4 个 w 比特的字组成。字长 w 与明文分组长度 n 的关系满足 $w = n/4$。设 r 为迭代轮数。FBC 算法主要参数见表 1。

表 2.1 算法主要参数取值

版本	分组长度 n	主密钥长度 m	字宽度 w	迭代轮数 r
FBC128-128	128	128	32	48
FBC128-256	128	256	32	64
FBC256-256	256	256	64	80

2.3 符号

本规范主要使用下面的一些符号：

P	n 比特明文
C	n 比特密文
k	m 比特主密钥
k_i	第 i 个子密钥字，i=0,1,…,2r+1
a_i, b_i, c_i, d_i	第 i 轮迭代的状态字，每个字的长度为 w 比特, i=0,1,2,…,r+1
u, v	长度为 w 比特的字

u_i, v_i	字 u 和 v 的第 i 个长度为 $w/4$ 比特的子块，i=1,2,3,4
$u_i[j]$, $v_i[j]$	子块 u_i 和 v_i 的第 j 比特，j=0,1,2,…,$w/4$-1
1	长度为 w 比特的全 1 字符块

和运算符：

⊕	面向比特的异或运算
&	面向比特的与运算
<<<	w 比特的字左循环移位运算
‖	字符串连接符

2.4 算法描述

2.4.1 算法结构

FBC 算法基于四路两重 Feistel 结构设计，轮函数 F 采用 bit-slice 模式构造，算法主体结构如图 2.1 所示。

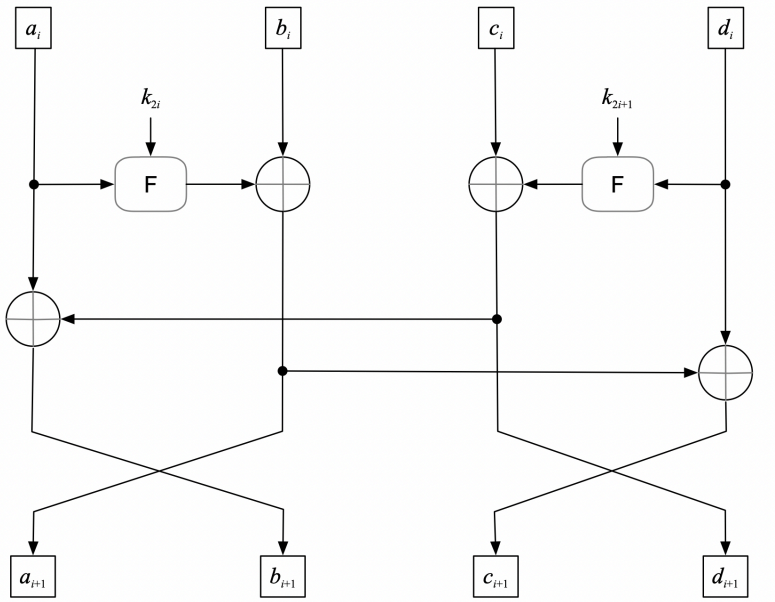

图 2.1 FBC 算法结构图

2.4.2 轮函数 F

轮函数F的输入包含2个w比特的字x和y，输出一个w比特的字z，记$z=\mathrm{F}(x,y)$。具体运算过程如下：

1) 子密钥加： $u = x \oplus y$。

2) 列变换：将 u 二进制表示分解成 4 个等宽的字符串，每个字符串包含 $w/4$ 个比特，即 $u = u_1 \| u_2 \| u_3 \| u_4$，令：

$$v_1[j] \| v_2[j] \| v_3[j] \| v_4[j] = S(u_1[j] \| u_2[j] \| u_3[j] \| u_4[j]), j=0,1,\ldots,w/4-1$$

这里 S 为 4 进 4 出的 S 盒变换，见表 2.2。记 $v = v_1 \| v_2 \| v_3 \| v_4$。

表 2.2 S 盒真值表

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	15	4	9	14	11	8	2	7	12	13	3	6	1	0

3) 行变换：$z = L_{s,t}(v) = v \oplus (v <<< s) \oplus (v <<< t)$

在行变换 $L_{s,t}$ 中参数 s, t 取值与 w 有关，见表 2.3。

表 2. 3 L 线性变换中循环参数 s, t 取值

w	32	64
s	3	17
t	10	58

2.4.3 密钥扩展算法 ExpandKey

针对 FBC128-128 和 FBC256-256，将 m 比特主密钥 k 分成 4 个 w 比特的字 k_0,k_1,k_2,k_3，对 $i=0,1,\ldots,2r-5$，计算

$$k_{i+4} = (k_i \oplus \mathbf{1}) \oplus (k_{i+1} \oplus \mathbf{1}) \,\&\, k_{i+2} \,\&\, k_{i+3} \oplus i,$$

$$k_{i+4} = L_{7,25}(k_{i+4}) = k_{i+4} \oplus (k_{i+4} \lll 13) \oplus (k_{i+4} \lll 22).$$

针对 FBC128-256，将 256 比特主密钥 k 分成 8 个 32 比特的字 $k_0,k_1,k_2,\ldots,k_7$，对 $i=0,1,\ldots,2r-9$，计算

$$k_{i+8} = (k_i \oplus \mathbf{1}) \oplus (k_{i+1} \oplus \mathbf{1}) \,\&\, k_{i+2} \,\&\, k_{i+3} \oplus i,$$

$$k_{i+8} = L_{7,25}(k_{i+8}) = k_{i+8} \oplus (k_{i+8} \lll 13) \oplus (k_{i+8} \lll 22).$$

2.4.4 加密过程

1) 首先根据 4.3 节密钥扩展算法将主密钥 k 扩展成子密钥序列 k_i，$i=0,1,\ldots,2r-1$。

2) 将输入明文块 P 分成 4 个长度为 w 比特的子块，即 $P = a_0 \| b_0 \| c_0 \| d_0$。重复执行下列操作 r-1 次(RoundFunc)：

   a) $a_{i+1} = F(a_i, k_{2i}) \oplus b_i$

   b) $c_{i+1} = a_{i+1} \oplus d_i$

   c) $d_{i+1} = F(d_i, k_{2i+1}) \oplus c_i$

d) $b_{i+1} = d_{i+1} \oplus a_i$

3) 然后执行下列操作 1 次(FinalRoundFunc)(不交换顺序)：

a) $b_{r+1} = \mathrm{F}(a_r, k_{2r-2}) \oplus b_r$

b) $d_{r+1} = b_{r+1} \oplus d_r$

c) $c_{r+1} = \mathrm{F}(d_r, k_{2r-1}) \oplus c_r$

d) $a_{r+1} = c_{r+1} \oplus a_r$

4) 最后令输出密文块 $\mathrm{C} = a_{r+1} \| b_{r+1} \| c_{r+1} \| d_{r+1}$。

2.4.5 解密过程

1) 首先根据密钥扩展算法将主密钥 k 扩展成子密钥序列 k_i，i=0,1,…,2r-1。

2) 将输入密文块 C 分成 4 个长度为 w 比特的子块，即 $\mathrm{C} = a_0 \| b_0 \| c_0 \| d_0$。首先执行下列操作 1 次：

a) $a_1 = a_0 \oplus c_0$

b) $b_1 = \mathrm{F}(a_1, k_{2r-2}) \oplus b_0$

c) $d_1 = b_0 \oplus d_0$

d) $c_1 = \mathrm{F}(d_1, k_{2\mathrm{r}-1}) \oplus c_0$

3) 然后对 i=1,2,…,r，重复执行下列操作 r-1 次：

a) $a_{i+1} = b_i \oplus d_i$

b) $b_{i+1} = \mathrm{F}(a_{i+1}, k_{2(r-i)-2}) \oplus a_i$

c) $d_{i+1} = a_i \oplus c_i$

d) $c_{i+1} = \mathrm{F}(d_{i+1}, k_{2(\mathrm{r}-i)-1}) \oplus d_i$

4) 最后令输出密文块 $\mathrm{P} = a_{r+1} \| b_{r+1} \| c_{r+1} \| d_{r+1}$。

3 设计准则

本章给出了 FBC 算法的一些设计准则，包括整体设计准则和部件设计准则两大类。

3.1 整体设计准则

FBC 算法整体上遵循简单、简洁、轻量、高效、安全等准则设计。其主要体现于：

1) 简单

FBC 算法基础运算部件简单，仅仅由通用部件异或、循环移位、逻辑与和 S 盒等构造。整个加解密过程和密钥扩展过程均简单明了。

2) 简洁

FBC 算法结构简洁，整体采用四路两重 Feistel 结构设计，F 函数基于 bit-slice 技术构造，整个算法没有任何冗余操作。

3) 轻量

FBC 算法部件采用轻量化设计，其中 S 盒基于 NFSR 构造，硬件实现仅需 5 个非门、4 个或门和 4 个与门。线性变换 L 每个输出比特仅需 2 个或门。密钥扩展同样采用轻量化设计，每轮子密钥比特硬件实现仅需 2 个非门，4 个异或门和 2 个与门。

4) 高效

FBC 算法具有多种高效的软硬件实现方法。在软件软件实现，可以采用 4 进 w 出或者 8 进 w 出表实现，亦可采用无查表快速逻辑实现方法进行实现；在硬件实现方面，局部可以基于 NFSR 对 S 盒和密钥扩展高效实现，整体既可以同时实现左右两路 Feistel 结构也可以只实现一侧 Feistel 结构。

5) 安全

FBC 算法整体采用四路两重 Feistel 结构，F 函数采用 bit-slice 技术构造，局部属于 SPN 结构，其中 S 盒选择的是密码学性质最优的 S 盒，置换 P 选择差分/线性分支数为 4 的线性变换，在结构上没有弱点。只要迭代足够的轮数，整个算法就具有较高的安全性。

3.2 部件设计准则

3.2.1 S 盒

S 盒在设计时主要遵循了如下准则：

1) 4 进 4 出的置换

2) 具有最优的差分均匀度和非线性度

3) 至少有一个分量布尔函数代数次数为 3

4) 没有不动点

5) 轻量化

6) 具有高效的软件实现

7) 在上述条件下硬件实现代价最低

根据上述准则，我们选择基于 NFSR 来构造最优的 S 盒，方法如下：

首先定义 4 级 NFSR 如下：

$$f(x_0, x_1, x_2, x_3) = 1 \oplus x_0 \oplus x_3 \oplus x_2 x_3$$

设由 f 定义的 NFSR 生成的序列的前八项为：

$$x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7$$

令

$$x = x_0 x_1 x_2 x_3$$

和

$$y = x_4 x_5 x_6 x_7$$

则 NFSR 诱导出F_2^4到F_2^4上的一个置换 S: $y = S(x)$。

3.2.2 线性变换 L

线性变换 L 在设计时主要遵循了如下准则：

1）w 位进 w 位出的线性置换

2）差分分支数和线性分支数不低于 4

3）具有最高的周期

4）具有最少个数的不动点

5）简单

6）轻量化

根据上述准则，我们选择如下仅由异或和循环移位两种运算构成的线性变换：

$$L_{s,t}(v) = v \oplus (v <<< s) \oplus (v <<< t),$$

其中其中参数 s, t 取值与 w 有关，见第二章表 3。

3.2.3 密钥扩展

密钥扩展在设计时主要遵循了如下准则：

1）非线性置换

2）简单

3）轻量化

4）高效软硬件实现

5）能抵抗滑动攻击

根据上述准则，我们采用类似 bit-slice 技术设计了密钥扩展，在纵向列变换上采用M序列构造方法，在横向行变换上面采用线性变换L，在满足轻量化需求下达到快速混淆和扩散的目的。

4 部件分析

4.1.1 S 盒

定义 4.1（代数次数） 设正整数 n。一个 $n\times n$ 的 S 盒是 F_2^n 到 F_2^n 上的一个映射：$s=(f_1,f_2,\cdots,f_n)$，这里 $f_i(1\le i\le n)$ 均是 n 元布尔函数。S 盒 s 的代数次数为其分量函数的线性组合的最低代数次数，即：

$$\deg(s)=\min\{\deg(c_1f_1+c_2f_2+\cdots+c_nf_n)\,|\,(c_1,c_2,\cdots,c_n)\neq 0\}。$$

定义 4.2（非线性度） 对于一个 $n\times n$ 的 S 盒 $y=s(x)$，定义

$$NL(s)=2^{n-1}-\frac{1}{2}\max_{v\neq 0,v\in F_2^n,w\in F_2^n}\left|\sum_{x\in F_2^n}(-1)^{\langle v,F(x)\rangle+\langle w,x\rangle}\right|,$$

为该 S 盒 s 的非线性度。

定义 4.3（差分均匀度） 对于一个 $n\times n$ 的 S 盒 $y=s(x)$，定义

$$\max_{\Delta x\neq 0,\Delta x,\Delta y\in F_2^n}|\{x\,|\,S(x+\Delta x)+S(x)=\Delta y\}|$$

为该 S 盒 s 的差分均匀度。

定义 4.4（图代数免疫度） 对于一个 $n\times n$ 的 S 盒 $y=s(x)$，定义

$$\min\{\deg(F)\,|\,F\in B_{2n},F(x,y)=0,F\neq 0,\forall x\in F_2^n,y=S(x)\}$$

为该 S 盒 s 的图代数免疫度。

定义 4.5（不动点） 对于一个 $n\times n$ 的 S 盒 $y=s(x)$，称满足 s(x)= x 的解为 S 盒的不动点。

对于 FBC 算法族选择的 4 进 4 出的 S 盒的密码性质见表 4.1，其差分均匀性、非线性度、图代数免疫度都达到了最优，没有不动点。

表 4.1 S 盒密码学指标

差分均匀性	非线性度	图代数免疫度	代数次数	不动点个数
4	4	2	2	0

4.1.2 线性变换 L

定义 4.6（差分分指数） 有限域 F_q 上的一个变换 L 被称之为是 F_q 上的 n 维线性变换，是指其对任意 $X,Y \in F_q^{\,n}, c \in F_q$ ，均满足

$$L(X+Y)=L(X)+L(Y),\ L(cX)=cL(X)。$$

对一个线性变换 $\mathrm{L}: F_q^{\,n} \to F_q^{\,n}$ ，$Y=L(X)$ ，可以用矩阵的形式表示，即存在 F_q 上的一个 $n\times n$ 的矩阵 M ，使得对任意 X ，有 $Y=L(X)=MX$ 。线性变换 L 的差分分指数定义为

$$d_b=\min_{X\neq 0}(w_H(X)+w_H(MX))\ ，$$

其中 $w_H(X)$ 是 X 的汉明重量，也就是非零分量的个数。

由定义可知，一个线性变换 L 的差分分指数总是小于等于 $n+1$。

定义 4.7（线性分指数） 对线性变换 $Y=L(X)=MX$ ，其线性分指数定义为

$$l_b=\min_{0\neq\alpha\in F_q^n}(w_H(\alpha)+w_H(M'\alpha))\ ，$$

这里 M' 表示矩阵 M 的转置。

定义 4.8 (周期) 对任意给定的非零变换 P 和正整数 k,定义 $P^{(k)}(x)=P(P^{(k-1)}(x))$，$P^{(1)}(x)=P(x)$，$P^{(0)}(x)=x$。称使得对所有 x 均有 $P(t)(x)=x$ 成立的最小正整数为变换 P 的周期。

对于 FBC 算法族选择的线性变换 $L_{s,t}$，其密码性质如下：

表 4.2 线性变换 $L_{s,t}$ 的密码学指标

差分分支数	线性分支数	周期	不动点个数
4	4	w	2

5 安全性评估

5.1 差分分析

对 FBC 算法族，我们采用混合整数线性规划(MILP)方法搜索了差分分析中最优活动S盒个数，对于r轮迭代，其最优的活动S盒个数为2r。对FBC128-128、FBC256-256达到安全界的迭代轮数为32和64，而实际算法迭代轮数为48和80，我们认为它们可以完全抵抗差分类密码分析；对FBC128-256，迭代64轮时活动S盒个数为128，刚好达到安全界。考虑到分组密码剥离轮技术的应用，MILP活动S盒个数会低于64。但是在实际差分密码分析中，真实差分路径的活动 S 盒个数往往会大于 MILP 方法得到的理论估计，且并不是每个活动 S 盒的取值概率都取到最大值，因此我们相信FBC128-256对于差分分析来说依然是安全的，只是没有像FBC128-128和FBC256-256的安全余量那么大。

下面以FBC128-128为例介绍MILP具体分析过程。

设F函数的输入记作(x,k)，这里k表示相应的轮密钥。设$x=(x_0,x_1,\cdots,x_{31})\in\{0,1\}^{32}$，将其写成如下形式：

$$x_0,\ x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6,\ x_7,$$
$$x_8,\ x_9,\ x_{10},x_{11},x_{12},x_{13},x_{14},x_{15},$$
$$x_{16},x_{17},x_{18},x_{19},x_{20},x_{21},x_{22},x_{23},$$
$$x_{24},x_{25},x_{26},x_{27},x_{28},x_{28},x_{30},x_{31}$$

注意到线性变换$L(x)=x\oplus(x\lll 3)\oplus(x\lll 10)$，$x$的每一列经过$L$变换后仍然在同一列中。引入二元变量组$(a_0,a_1,\cdots,a_7)$，分别对应$x$的每一列，即每个 S 盒，$a_i$刻画第列的活动S盒，即当该列为零时，即对应的S盒输入没有差分a_i取值为0；否则取值为1。接下来我们对轮函数用线性不等式刻画，将最少活动S盒个数作为目标函数并构造相应的MILP问题，最后用Gurobi求解目标值。

(1) 对函数 F 的差分模式的不等式刻画

设$(a_0,a_1,\cdots,a_7)$表示 F 函数的输入差分模式，$(b_0,b_1,\cdots,b_7)$表示 F 函数的输出差分模式。为了刻画$L(x)=x\oplus(x\lll 3)\oplus(x\lll 10)$变换的输入输出差分模式，我们穷尽$L$变换的所有输入，最后我们发现输入输出差分重量之和为4的只有八种可能，见表5.1。

表 5.1 输入输出差分重量之和为 4 的差分模式

1000000011010000	0100000001101000
0010000000110100	0001000000011010
0000100000001101	0000010010000110
0000001001000011	0000000110100001

而其余情况其重量或者为零或者大于等于 6。于是我们用线性不等式刻画如下：

$$\begin{aligned}
a_0+a_1+a_2+a_3+a_4+a_5+a_6+a_7 &= d_0\\
b_0+b_1+b_2+b_3+b_4+b_5+b_6+b_7 &= 3d_0\\
a_0+a_1+\cdots+a_7+b_0+b_1+\cdots+b_7 &\geq 6d_1\\
d_0+d_1 &= d_2\\
a_0+a_1+a_2+a_3+a_4+a_5+a_6+a_7 &\geq d_2\\
b_0+b_1+b_2+b_3+b_4+b_5+b_6+b_7 &\geq d_2\\
d_2 &\geq a_0\\
d_2 &\geq a_0\\
&\vdots\\
d_2 &\geq a_7\\
d_2 &\geq b_0\\
d_2 &\geq b_1\\
&\vdots\\
d_2 &\geq b_7
\end{aligned}$$

这里d_0, d_1和d_2为引入的辅助变量，这三组变量的取值均为$\{0,1\}$.

(2) 对轮函数的差分模式的不等式刻画：

设第i轮函数的输入记作$\left(in_0^{(i)}, in_1^{(i)}, in_2^{(i)}, in_3^{(i)}\right)$，输出记作$\left(out_0^{(i)}, out_1^{(i)}, out_2^{(i)}, out_3^{(i)}\right)$，则有

$$out_0^{(i)} = F\left(in_0^{(i)}\right) \oplus in_1^{(i)}, out_1^{(i)} = in_0^{(i)} \oplus out_3^{(i)},$$

$$out_2^{(i)} = out_0^{(i)} \oplus in_3^{(i)}, out_3^{(i)} = F\left(in_3^{(i)}\right) \oplus in_2^{(i)}$$

不妨设第i轮的输入差分模式记作$\left(v_0^{(i)}, v_1^{(i)}, \cdots, v_{31}^{(i)}\right)$，输出差分模式记作$\left(v_0^{(i+1)}, v_1^{(i+1)}, \cdots, v_{31}^{(i+1)}\right)$，即第$i+1$轮的输入差分模式，则$\left(v_0^{(i)}, v_1^{(i)}, \cdots, v_7^{(i)}\right), \left(v_8^{(i)}, v_9^{(i)}, \cdots, v_{15}^{(i)}\right), \left(v_{16}^{(i)}, v_{17}^{(i)}, \cdots, v_{23}^{(i)}\right)$和$\left(v_{24}^{(i)}, v_{25}^{(i)}, \cdots, v_{31}^{(i)}\right)$分别对应四个输入$in_0^{(i)}, in_1^{(i)}, in_2^{(i)}$和$in_3^{(i)}$的差分模式，而

$\left(v_0^{(i+1)}, v_1^{(i+1)}, \cdots, v_7^{(i+1)}\right), \left(v_8^{(i+1)}, v_9^{(i+1)}, \cdots, v_{15}^{(i+1)}\right), \left(v_{16}^{(i+1)}, v_{17}^{(i+1)}, \cdots, v_{23}^{(i+1)}\right)$ 和 $\left(v_{24}^{(i+1)}, v_{25}^{(i+1)}, \cdots, v_{31}^{(i+1)}\right)$分别对应四个输出$out_0^{(i)}, out_1^{(i)}, out_2^{(i)}$和$out_3^{(i)}$的差分模式.

引入中间变量$\left(m_0^{(i)}, m_1^{(i)}, \cdots, m_{15}^{(i)}\right)$，即两个 F 函数的输出差分模式，则根据 F 函数的不等式刻画需要对 $\left\{\left(v_0^{(i)}, v_1^{(i)}, \cdots, v_7^{(i)}\right), \left(m_0^{(i)}, m_1^{(i)}, \cdots, m_7^{(i)}\right)\right\}$ 和 $\left\{\left(v_{24}^{(i)}, v_{25}^{(i)}, \cdots, v_{31}^{(i)}\right), \left(m_8^{(i)}, m_9^{(i)}, \cdots, m_{15}^{(i)}\right)\right\}$这两对输入输出差分模式做相应的不等式刻画。对于$\left\{\left(m_0^{(i)}, m_1^{(i)}, \cdots, m_7^{(i)}\right), \left(v_8^{(i)}, v_9^{(i)}, \cdots, v_{15}^{(i)}\right), \left(v_0^{(i+1)}, v_1^{(i+1)}, \cdots, v_7^{(i+1)}\right)\right\}$，需要做出如下的不等式刻画：

$$m_j^{(i)} + v_{j+8}^{(i)} + v_j^{(i+1)} \geq 2d_j^{(i)}$$
$$d_j^{(i)} \geq m_j^{(i)}$$
$$d_j^{(i)} \geq v_{j+8}^{(i)}$$
$$d_j^{(i)} \geq v_j^{(i+1)}$$

这里$j = 0,1,\cdots,7$，$d_j^{(i)}$为辅助变量，取值为$\{0,1\}$。

同理需要对

$$\left\{\left(v_8^{(i+1)}, v_9^{(i+1)}, \cdots, v_{15}^{(i+1)}\right), \left(v_0^{(i)}, v_1^{(i)}, \cdots, v_7^{(i)}\right), \left(v_{24}^{(i+1)}, v_{25}^{(i+1)}, \cdots, v_{31}^{(i+1)}\right)\right\},$$
$$\left\{\left(v_{16}^{(i+1)}, v_{17}^{(i+1)}, \cdots, v_{23}^{(i+1)}\right), \left(v_0^{(i+1)}, v_1^{(i+1)}, \cdots, v_7^{(i+1)}\right), \left(v_{24}^{(i)}, v_{25}^{(i)}, \cdots, v_{31}^{(i)}\right)\right\}$$
$$\left\{\left(v_{24}^{(i+1)}, v_{25}^{(i+1)}, \cdots, v_{31}^{(i+1)}\right), \left(m_8^{(i)}, m_9^{(i)}, \cdots, m_{15}^{(i)}\right), \left(v_{16}^{(i)}, v_{17}^{(i)}, \cdots, v_{23}^{(i)}\right)\right\}$$

做相应的不等式刻画。

综上每一轮引入 16 个临时变量$\left(m_0^{(i)}, m_1^{(i)}, \cdots, m_{15}^{(i)}\right)$以及 38 个辅助变量。

(3) 限制条件刻画

首先，设初始输入差分模式记作$\left(v_0^{(0)}, v_1^{(0)}, \cdots, v_{31}^{(0)}\right)$，则有

$$v_0^{(0)} + v_1^{(0)} + \cdots + v_{31}^{(0)} \geq 1$$

该不等式表示输入差分不为零。

其次，所有变量取值范围均为$\{0,1\}$。

(4) 目标函数刻画

对于每一轮的输入差分模式$\left(v_0^{(i)}, v_1^{(i)}, \cdots, v_{31}^{(i)}\right)$，活跃 S 盒的体现在$\left(v_0^{(i)}, v_1^{(i)}, \cdots, v_7^{(i)}\right)$和$\left(v_{24}^{(i)}, v_{25}^{(i)}, \cdots, v_{31}^{(i)}\right)$，即两个 F 函数的输入差分模式。因此目标函数为

$$min \sum_{i=0}^{n-1}\left(v_0^{(i)} + v_1^{(i)} + \cdots, v_7^{(i)}\right) + \left(v_{24}^{(i)} + v_{25}^{(i)} + \cdots + v_{31}^{(i)}\right)$$

这里n表示轮数。

我们利用 Gurobi 运行该 MILP 问题，当轮数为 32 时，活动 S 盒的个数为 64；当轮数为 64 时，活动 S 盒的个数为 128。

5.2 线性分析

对 FBC 算法族，我们采用混合整数线性规划(MILP)方法搜索了线性分析中最优活动 S 盒个数，对于 r 轮迭代，其最优的活动 S 盒个数为 2r。对 FBC128-128、FBC128-256、FBC256-256 达到安全界的迭代轮数为 32、64 和 64，而实际算法迭代轮数为 48、64 和 80，我们认为它们可以完全抵抗线性类密码分析。

下面以 FBC128-128 为例给出了其 MILP 分析过程。

设$x = (x_0, x_1, \cdots, x_{31}) \in \{0,1\}^{32}$表示$F$函数的输入，类似差分分析，引入二元变量组$(a_0, a_1, \cdots, a_7)$，分别对应$x$的每一列，即每个 S 盒。$a_i$刻画第$i$列的活动 S 盒，即当该列为零时，即对应 S 盒的输入线性掩码为 0，a_i取值为 0；否则取值为 1. 然后我们对轮函数用线性不等式刻画，将最少活动 S 盒个数作为目标函数并构造相应的 MILP 问题，最后用 Gurobi 求解目标值。

(1) 对 F 函数的线性掩码模式的不等式刻画

设$(a_0, a_1, \cdots, a_7)$表示 F 函数的输入线性掩码模式，$(b_0, b_1, \cdots, b_7)$表示 F 函数的输出线性掩码模式。根据线性掩码与差分的对偶性，此时我们穷尽线性变换$L^T(x) = x \oplus (x \lll 3) \oplus (x \lll 10)$（该线性变换为$L(x)$的转置）的所有输入，最后我们发现输入输出线性掩码重量为 4 的只有八种可能，见表 5.2。

表 5.2 输入输出重量之和为 4 的线性掩码

1000000010100001	0100000011010000
0010000001101000	0001000000110100
0000100000011010	0000010000001101

0000001010000110	0000000101000011

而其余情况其重量或者为零或者大于等于 6。我们用线性不等式刻画如下：

$$\begin{aligned}
a_0+a_1+a_2+a_3+a_4+a_5+a_6+a_7 &= d_0\\
b_0+b_1+b_2+b_3+b_4+b_5+b_6+b_7 &= 3d_0\\
a_0+a_1+\cdots+a_7+b_0+b_1+\cdots+b_7 &\geq 6d_1\\
d_0+d_1 &= d_2\\
a_0+a_1+a_2+a_3+a_4+a_5+a_6+a_7 &\geq d_2\\
b_0+b_1+b_2+b_3+b_4+b_5+b_6+b_7 &\geq d_2\\
d_2 &\geq a_0\\
d_2 &\geq a_0\\
&\vdots\\
d_2 &\geq a_7\\
d_2 &\geq b_0\\
d_2 &\geq b_1\\
&\vdots\\
d_2 &\geq b_7
\end{aligned}$$

这里d_0, d_1和d_2为引入的辅助变量，这三组变量的取值均为$\{0,1\}$。

(2) 对轮函数的线性掩码模式的不等式刻画

设第i轮函数的输入记作$\left(in_0^{(i)}, in_1^{(i)}, in_2^{(i)}, in_3^{(i)}\right)$，输出记作$\left(out_0^{(i)}, out_1^{(i)}, out_2^{(i)}, out_3^{(i)}\right)$，则有

$$\begin{aligned}
out_0^{(i)} &= F\left(in_0^{(i)}\right) \oplus in_1^{(i)}, out_1^{(i)} = in_0^{(i)} \oplus out_3^{(i)},\\
out_2^{(i)} &= out_0^{(i)} \oplus in_3^{(i)}, out_3^{(i)} = F\left(in_3^{(i)}\right) \oplus in_2^{(i)}
\end{aligned}$$

不妨设第i轮的输入线性掩码模式记作$\left(v_0^{(i)}, v_1^{(i)}, \cdots, v_{31}^{(i)}\right)$，输出线性掩码模式记作$\left(v_0^{(i+1)}, v_1^{(i+1)}, \cdots, v_{31}^{(i+1)}\right)$，即第$i+1$轮的输入线性掩码模式，引入中间变量$\left(m_0^{(i)}, m_1^{(i)}, \cdots, m_{15}^{(i)}\right)$，即两个 F 函数的输出线性掩码模式，则根据 F 函数的不等式刻画需要对$\left\{\left(v_8^{(i)}, v_9^{(i)}, \cdots, v_{15}^{(i)}\right), \left(m_0^{(i)}, m_1^{(i)}, \cdots, m_7^{(i)}\right)\right\}$和$\left\{\left(v_{16}^{(i)}, v_{17}^{(i)}, \cdots, v_{23}^{(i)}\right), \left(m_8^{(i)}, m_9^{(i)}, \cdots, m_{15}^{(i)}\right)\right\}$这两对输入输出线性掩码模式做相应的不等式刻画。

对于线性掩码模式组$\left\{\left(v_8^{(i)},v_9^{(i)},\cdots,v_{15}^{(i)}\right),\left(v_{16}^{(i+1)},v_{17}^{(i+1)},\cdots,v_{23}^{(i+1)}\right),\left(v_0^{(i+1)},v_1^{(i+1)},\cdots,v_7^{(i+1)}\right)\right\}$，需要做出如下的不等式刻画：

$$v_{j+8}^{(i)}+v_{j+16}^{(i+1)}+v_j^{(i+1)}\geq 2d_j^{(i)}$$
$$d_j^{(i)}\geq v_{j+8}^{(i)}$$
$$d_j^{(i)}\geq v_{j+16}^{(i+1)}$$
$$d_j^{(i)}\geq v_j^{(i+1)}$$

这里$j=0,1,\cdots,7$，$d_j^{(i)}$为辅助变量，取值为$\{0,1\}$.

同理需要对

$$\left\{\left(m_0^{(i)},m_1^{(i)},\cdots,m_7^{(i)}\right),\left(v_0^{(i)},v_1^{(i)},\cdots,v_7^{(i)}\right),\left(v_8^{(i+1)},v_9^{(i+1)},\cdots,v_{15}^{(i+1)}\right)\right\},$$
$$\left\{\left(m_8^{(i)},m_9^{(i)},\cdots,m_{15}^{(i)}\right),\left(v_{24}^{(i)},v_{25}^{(i)},\cdots,v_{31}^{(i)}\right),\left(v_{16}^{(i+1)},v_{17}^{(i+1)},\cdots,v_{23}^{(i+1)}\right)\right\}$$
$$\left\{\left(v_0^{(i+1)},v_1^{(i+1)},\cdots,v_7^{(i+1)}\right),\left(v_{16}^{(i)},v_{17}^{(i)},\cdots,v_{23}^{(i)}\right),\left(v_{24}^{(i+1)},v_{25}^{(i+1)},\cdots,v_{31}^{(i+1)}\right)\right\}$$

做相应的不等式刻画。

综上每一轮引入 16 个临时变量$\left(m_0^{(i)},m_1^{(i)},\cdots,m_{15}^{(i)}\right)$以及 38 个辅助变量。

(3) 限制条件刻画

首先，设初始输入线性掩码模式记作$\left(v_0^{(0)},v_1^{(0)},\cdots,v_{31}^{(0)}\right)$，则有

$$v_0^{(0)}+v_1^{(0)}+\cdots+v_{31}^{(0)}\geq 1$$

该不等式表示输入线性掩码不为零。

其次，所有变量取值范围均为$\{0,1\}$。

(4) 目标函数刻画

对于每一轮的输入线性掩码模式$\left(v_0^{(i)},v_1^{(i)},\cdots,v_{31}^{(i)}\right)$，活跃 S 盒的体现在$\left(v_8^{(i)},v_9^{(i)},\cdots,v_{15}^{(i)}\right)$和$\left(v_{16}^{(i)},v_{17}^{(i)},\cdots,v_{23}^{(i)}\right)$，即两个 F 函数的输出线性掩码模式。因此目标函数为

$$min\sum_{i=0}^{n-1}\left(v_8^{(i)}+v_9^{(i)}+\cdots+v_{15}^{(i)}\right)+\left(v_{16}^{(i)}+v_{17}^{(i)}+\cdots+v_{23}^{(i)}\right)$$

这里n表示轮数。

事实上，线性掩码模式的不等式刻画与差分模式的不等式刻画完全等价，我们利用 Gurobi 运行该 MILP 问题，当轮数为 32 时，活动 S 盒的个数为 64；当轮数为 64 时，活动 S 盒的个数为 128。

5.3 不可能差分分析

对于不可能差分分析，我们同样利用混合整数线性规划 MILP 来搜索 FBC 最长轮数的不可能差分模式，具体的转化 MILP 不等式方法与 5.1 节类似，不再赘婿。我们在具体分析中构建了三种混合整数线性规划 MILP 模型，在有限的计算资源下，可得到 FBC128-128 和 FBC128-256 的 7 轮不可能差分和 FBC256-256 的 13 轮不可能差分。下面给出三种模型的构造方法，由已知的不可能差分分析结果来看，几乎所有的不可能差分路径的输入和输出差分模式分别只有一个活跃 s 盒，因此，三种模型中，选定输入输出差分时，应满足该输入（输出）只能使单个 s 盒活跃。

Model1：变量基于比特，将第一轮的输入差分和最后一轮的输出差分信息（α，β）添加到搜索 i（i从第一轮开始遍历）轮差分路径的模型中，去掉目标函数，判断该模型有无解即可。若 i 轮模型无解，说明存在 i 轮的不可能差分路径（α，β），继续搜索 i+1 轮；若 i+1 轮有解，不再遍历 i+1 轮之后的模型，此时不可能差分路径(α，β)轮数最长为 i。遍历输入输出差分(α，β)，重复上面过程，则可得到最大轮数的不可能差分和对应的差分选取模式。

Model2：用到 Sasaki 和 Todo[1]提出的方法，同 Model1 类似，不同在于对 s 盒的刻画。这里对于 s 盒来说，任意的非 0 输入差分经过 s 盒都可以得到任意的非 0 输出差分，因此 s 盒的刻画只需要两个不等式，而且遍历差分模式时，只需要对活跃 s 盒的位置进行遍历，s 盒的输入输出任意取值即可。搜索过程与 Model1 相同。

Model3：将 s 盒作为变量，则只需刻画轮函数的线性层即可，相对于上述两种模型，变量减少大概 4 倍。遍历输入输出差分模式时，遍历活跃 s 盒的位置即可，搜索过程同上。

上述三种方法各有优略。Model1 遍历所有的输入输出差分，即使规定活跃 s 盒个数为 1，要搜索的模型个数也是远远大于其他两种，因此搜索时间最长；当轮数较大时，Model3 能快速给出模型是否有解，效率上 Model3>Model2>Model1；但由于 Model1 遍历所有输入输出差分，Model2 取定 s 盒的某个输入输出，Model3 只是遍历 s 盒位置，因此在轮数大小上 Model1>Model2>Model3。

在 FBC128 实验中，对于 Model1 和 Model2，由于计算能力有限，第八轮就已经不能计算出结果，第七轮无解；Model3 虽然可判断的轮数大，但是 Model3 在第七轮之后的所有轮数都有解。三个模型给出的结论一样，目前只能搜索到 7 轮不可能差分。

表 5.3 FBC 算法族不可能差分分析结果

	Model1	**Model2**	**Model3**
FBC128-128 和 FBC128-256	目前搜出 7 轮，8 轮单个模型搜索时间超过 34h，无结果无法判断	同 Model1	最长不可能差分 7 轮，超过 7 轮的所有模型均有解
FBC256-256	同 Model2	目前搜索到最长不可能差分 13 轮，14 轮单个模型搜索时间超过 13h，无结果	最长不可能差分 13 轮，超过 13 轮的所有模型均有解

5.4 积分分析

可分性（division property）是目前分析密码算法积分性质最有力的工具，并且可以借助混合整数线性规划的方法方便快速地搜索算法的积分区分器。借助可分性可以搜索得到 FBC128-128 和 FBC128-256 算法的 7 轮积分区分器，受限于计算能力，目前还无法断定 FBC128-128 和 FBC128-256 是否存在 8 轮积分区分器。FBC256-256 可以搜索得到 10 轮积分区分器，但是目前同样无法断 FBC256-256 是否存在 11 轮积分区分器。

5.5 零相关分析

类似于不可能差分分析，我们利用混合整数线性规划搜索 FBC 算法族零相关的最长轮数。构造模型的方法同不可能差分分析的 Model3，区别在于对轮函数线性层描述与差分分析的相反，但实验结果同不可能差分结果一样，对于 FBC128-128 和 FBC128-256 的零相关分析，目前能找到的最大轮数为 7。对于 FBC256-256 的零相关分析，目前能找到的最大轮数为 13。

6 软硬件性能评估

6.1 软件性能评估

FBC 算法族具有灵活高效的多种软件实现方式，包括无查表实现、4 进 w 出查表实现、8 进 w 出查表实现，在内存容许的情况下，甚至可以做成 16 进 w 出的查大表实现。

6.1.1 无查表实现

函数 F 可以通过无查表快速实现，实现方式如下：

u = x ⊕ k;

u0 = u & bitmask_const;

u1 = (u >> w/4) & bitmask_const;

u2 = (u >> w/2) & bitmask_const;

u3 = (u >> (3*w/4)) & bitmask_const;

v2 = u2 ⊕ bitmask_const;

u4 = (u1 ⊕ bitmask_const) ⊕ u3 & v2;

v3 = u3 ⊕ bitmask_const;

u5 = v2 ⊕ u4 & v3;

v4 = u4 ⊕ bitmask_const;

u6 = v3 ⊕ u5 & v4;

v5 = u5 ⊕ bitmask_const;

u7 = v4 ⊕ u6 & v5;

y = v4 | (v5 << w/4) | (v6 << w/2) | (v7 << (3*w/4));

注：这里 bitmask_const 为比特掩码常量，其为宽度为 w/4 的全 1 字符串。

从上述快速实现方式中，可以看出，函数F可以通过5个异或，8个逻辑与运算，5 个非运算(与全 1 常量异或运算可以看成是非运算)，6 个移位运算，3 个或运算共 27 个基础运算就可以实现。

因此 FBC 算法族的一轮迭代只需要 27+4=31 个基础运算便可以实现。整个加密过程和解密过程只需要 31r 个基础运算就可以完成。

6.1.2 4 进 w 出小表实现

函数 F 的主体可以看成是 SPN 结构，因此可以采用类似 AES 查表实现方式。在内存限制的情况下，我们可以采用 4 进 w 出小表实现。

在采用 4 进 w 出小表方法时，我们有两种实现方式：一种是只在 F 函数内采用 4 进 w 出小表实现，此时，需要将进入函数 F 的输入 w 比特字改写为 w/4 个子块，然后对每个子块进行查表然后异或在一起。另外一种是在整个加密过程中调整比特位顺序，采用 4 比特块作为基础字单元来表示整个加密过程，在加密结束阶段重排比特位顺序组合成密文数据。后一种实现方式可以减少每轮迭代时对函数 F 的比特字的重排。

下面介绍后一种实现方式。首先将输入明文分成 4 个子块 a_i,b_i,c_i,d_i，并按照如下方式重排每个子块内的比特位顺序：

0	1	2	…	w/4-1
w/4	w/4+1	w/4+2	…	w/2-1
w/2	w/2+1	w/2+2	…	3w/4-1
3w/4	3w/4+1	3w/4+2	…	w-1

调整为：

0	4	8	…	w-4
1	5	9	…	w-3
2	6	10	…	w-2
3	7	11	…	w-1

其次，对每个 4 比特输入 x，建表

$$T_i[x] = L(0 \| 0 \| S(x) \| \cdots \| 0 \| 0)$$

注意在建第 i 张表时 S(x)恰好位于处于第 i 个 4 比特子块位置，其它子块为 0。这样对于宽度为 w 的字，我们需要建立 w/4 张表。当这些表建立好之后，我们可以采用查询 T_i 表实现函数 F：

$u = x \oplus k;$

$y = T_0[u \,\&\, 15] \oplus T_1[(u >> 4) \,\&\, 15] \oplus \ldots \oplus T_{w/4-1}[(u >> (w-4))];$

采用上述方式需要 w/4 个异或、w/4-1 移位以及 w/4-1 个与运算。此时实现一轮迭代总共需要 3w/4+2 个基础运算。

6.1.3 8 进 w 出小表实现

在内存允许的情况下我们一般可采用建立 8 进 w 出小表来实现。其实现方式与采用 4 进 w 出小表方式几乎相同。唯一的区别在于，建立表的输入宽度为 4 进 w 出方式的 2 倍。具体建表过程如下：

$$T_i[x] = L(0 \| 0 \| (S(x_1) \| S(x_2)) \| \ldots \| 0 \| 0)$$

这里 $x=x_1 \| x_2$，每个子块的宽度为 0。此时需要建立 w/8 个字表就可以了。当这些表建立好之后，我们可以采用查询 T_i 表实现函数 F：

$u = x \oplus k;$

$y = T_0[u \& 255] \oplus T_1[(u >> 8) \& 255] \oplus \ldots \oplus T_{w/8-1}[(u >> (w-8))];$

采用上述方式需要 w/8 个异或、w/8-1 移位以及 w/8-1 个与运算。此时实现一轮迭代总共需要 3w/8+2 个基础运算。

6.1.4 软件自测试结果

在 64bit Windows 以及 32bit ARM 环境下,对 FBC 算法的加解密速率进行了测试，结果取10^5次 CBC 模式运算测试结果的平均值为加/解密速率。

表 6.1 FBC 算法软件测试环境

测试对象	测试环境
64bit Windows 环境下的算法速率优化 C 实现	Intel(R) Core(TM) i7-4700 CPU 3.40GHz 主频，64 位操作系统 Win10，8GB 内存，X86-64 环境 编译器：visual studio 2019, x64 release
32bit ARM 环境下的算法速率优化 C 实现	基于 NXP MIMXRT1060-EVK 主板 (Cortex M7, 600MHz, internal SRAM 1M, 32K data cache, 32K instruction cache) 编译器：arm-none-eabi-gcc 7.3.1 20180622 release

表 6.2 FBC 算法软件自测试实现速率

算法软件实现类别		加密速率(Mbps)	解密速率(Mbps)
64bit Windows 环境下的算法速率优化 C 实现	128/128	293.21	306.61
	128/256	221.72	233.07
	256/256	339.08	350.65
32bit ARM 环境下的算法速率优化 C 实现	128/128	38.57	38.66
	128/256	29.24	27.95
	256/256	22.66	21.78

6.2 硬件性能评估

6.2.1 硬件实现概述

FBC 算法族所有版本均可以轻量化实现。正如 3.2.1 节所示，S 盒可以采用 NFSR 方式实现，此时需要 4 个与门，4 个异或门，5 个非门。而对于线性变换 L，每比特需要 2 个异或门。因此实现函数 F，需要 w 个与门，3w 个异或门和 5w/4 个非门。

对于 FBC 算法的轮函数可以采用多种方式实现。一种是同时实现四路两重 Feistel 结构，此时需要 2w 个与门，10w 个异或门和 5w/2 个非门。另外一种是只实现两路 Feistel 结构，另外两路采用重复调用即可。此时需要 w 个与门，5w 个异或门和 5w/4 个非门就可以了。当然在 Feistel 结构内部，我们还可以更小宽度来实现轮函数，此时需要的硬件门数更少。

从上面的讨论可知，FBC 算法的硬件实现代价非常低。

6.2.2 verilog 实现测试

对 FBC 算法进行了 Verilog 硬件仿真实现自测试，结果如下表所示：

6.3 FBC 算法 Verilog 测试环境

测试对象	测试环境
算法 Verilog 硬件仿真实现	基于 Xilinx 公司的 ISE Design Suite 14.7 软件工具和 Xilinx virtex 5 FPGA 芯片

表 6.4 FBC 算法 Verilog 硬件仿真测试

	自测试结果		
	128/128	128/256	256/256
运算周期	(48+1)*32=1568	(64+1)*32=2080	(80+1)*32=2592
时钟约束	5ns	5 ns	5ns
加密速率	522Mbps	394Mbps	632Mbps
解密速率	522Mbps	394Mbps	632Mbps
面积	12696LUTs	15400LUTs	26486 LUTs
加密吞面比	0.92%	0.75%	21.9%
解密吞面比	1.28%	1.06%	17.8%