密码学报 ISSN 2095-7025 CN 10-1195/TN
Journal of Cryptologic Research, 2019, 6(6): 713–726

E-mail: jcr@cacrnet.org.cn
http://www.jcr.cacrnet.org.cn
Tel/Fax: +86-10-82789618

分组密码专刊

分组密码算法 FESH*

贾珂婷[1], 董晓阳[2], 魏淙洺[2], 李 铮[3], 周海波[4], 丛天硕[2]

1. 清华大学 计算机科学与技术系, 北京 100084
2. 清华大学 高等研究院, 北京 100084
3. 北京工业大学 信息学部计算机学院, 北京 100124
4. 山东大学 网络空间安全学院, 青岛 266237

通信作者: 贾珂婷, E-mail: ktjia@tsinghua.edu.cn

摘 要: 我们提出了一个新的分组密码算法 FESH, 该算法支持 128 比特和 256 比特的分组, 密钥支持 128, 192, 256, 384 和 512 比特, 共 6 个版本. FESH 算法采用替换-置换网络 (SPN) 结构作为算法整体结构, 轮函数设计简洁. 类似于 AES 竞赛中的 Serpent 算法, FESH 采用比特切片方式. 然而采用了更安全高效的 4 位 S 盒以及基于 4 分支 Feistel 结构的扩散层. FESH 算法可以抵抗差分分析、线性分析等现有的分析并具有足够的安全冗余. FESH 算法灵活性强, 在软件处理平台和硬件平台上都具有较好的性能.

关键词: FESH; 分组密码; 设计; 比特切片

中图分类号: TP309.7 **文献标识码:** A DOI: 10.13868/j.cnki.jcr.000336

中文引用格式: 贾珂婷, 董晓阳, 魏淙洺, 李铮, 周海波, 丛天硕. 分组密码算法 FESH[J]. 密码学报, 2019, 6(6): 713–726.

英文引用格式: JIA K T, DONG X Y, WEI C M, LI Z, ZHOU H B, CONG T S. On the design of block cipher FESH[J]. Journal of Cryptologic Research, 2019, 6(6): 713–726.

On the Design of Block Cipher FESH

JIA Ke-Ting[1], DONG Xiao-Yang[2], WEI Cong-Ming[2], LI Zheng[3], ZHOU Hai-Bo[4], CONG Tian-Shuo[2]

1. Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
2. Institute for Advanced Study, Tsinghua University, Beijing 100084, China
3. College of Computer Science, Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China
4. School of Cyber Science and Technology, Shandong University, Qingdao 266237, China

Corresponding author: JIA Ke-Ting, E-mail: ktjia@tsinghua.edu.cn

* 基金项目: 国家重点研发计划 (2017YFA0303903, 2018YFA0704701, 2018YFB0803400); 国家密码发展基金 (MMJJ20170121, MMJJ20180101); 浙江省重点研发计划 (2017C01062)

Foundation: National Key Research and Development Program of China (2017YFA0303903, 2018YFA0704701, 2018YFB0803400); National Cryptography Development Fund (MMJJ20170121, MMJJ20180101); Key R&D Project of Zhejiang Province (2017C01062)

收稿日期: 2019-11-25 定稿日期: 2019-12-10

Abstract: A new block cipher FESH is designed, which is a cipher family comprising of 6 distinct block ciphers with 128 and 256 block sizes, supporting key lengths of 128, 192, 256, 384, and 512 respectively. FESH is a substitution-permutation network (SPN) iterating a simple round transformation. Similar to the Serpent cipher, FESH applies bit-slice mode. However, a more secure 4-bit S-box with a 4-branch Feistel structure linear layer is adopted. FESH achieves high security against known attacks such as differential cryptanalysis, linear cryptanalysis etc. FESH is flexible for efficient implementations in both hardware and software.

Key words: FESH; block cipher; design; bit-slice

1 引言

分组密码是一类重要的对称密码算法, 广泛应用于网络通信中提供保密服务, 同时也是构建其他密码算法或密码协议的密码元语. 分组密码主要有两种结构—(广义) Feistel 结构和替换-置换网络 (SPN) 结构. Feistel 结构加解密采用相同的结构, 加解密算法同时实现可以节省资源, 但是一般 Feistel 结构轮数比较多, 加解密效率相对低一些. 替换-置换网络 (SPN) 结构通过多次迭代轮函数实现, 轮函数包括混淆层和扩散层, 解密是加密运算的逆, 一般情况下加密和解密的部件是不同的. SPN 结构中, 一类是基于 S 盒和 MDS 码的 AES 类结构, 该类结构算法有利于对抗差分和线性分析, 并进行安全性证明, 算法有 AES [1]、Midori [2] 和 SKINNY [3] 等. 另一类是 "SL" 设计结构, 将分组划分为 $s \times L$ 的矩阵, 纵向查询 s 比特的 S 盒 S-box, 横向针对 L 比特的字进行线性变换, 这种结构一般采用 4 比特 S 盒以便于进行侧信道防护 [4], 同时 S 盒可以采用比特切片实现, 便于 8 位微处理器 MCU 上实现, 基于该类结构的算法有 Serpent [5]、Noekeon [6]、PRIDE [7]、Rectangle [8] 等. 还有一些其他的 SPN 结构, 如基于 ARX 实现的结构, 该类算法软件实现效率比较快, 但是硬件实现耗费资源较多.

S 盒是分组密码设计的一个关键部件, 国际上密码算法比较通用的是 8 比特和 4 比特的两种 S 盒. 8 比特的 S 盒硬件实现占用面积比较大, 一般需要 300 多个门; 4 比特的 S 盒硬件实现占用面积比较小, 一般 20–40 个门. Saarinen 等 [9] 对所有 4 比特的活性 S 盒进行分析, 证明差分特征、线性特征和代数次数达最优的 S 盒共 16 个仿射等价类, 并对这些等价类的差分分支数 (BN) 进行了讨论. Serpent 和 PRESENT [10] 都采用了 BN = 3 的 S 盒的等价类, 但是这些算法并没有考虑低汉明重量的线性特征, 因此这两个算法更容易受到线性分析. 张文涛等在 RECTANGLE [8] 算法的设计中考虑了低汉明重量差分和线性的情况, 采用了 BN = 2 的 S 盒等价类.

扩散层是分组密码产生强雪崩的一个重要部件, 在 SPN 结构中扩散层及其逆运算都要便于实现. MDS 码是设计扩散层的方法之一, 如 AES、SKINNY 等, 这样算法便于对差分和线性分析进行活性盒估计, 提供最优的活性盒数, 但是 MDS 码实现效率较低. 比特置换是设计扩散层的另一种方法, 如 Rectangle、PRESENT 等, 这类算法扩散性较慢, 加解密需要的轮数比较多, 软件实现效率相对慢一些, 但是硬件实现占用面积较小; 扩散层也有采用基于字的混合运算的, 如 Serpent、Noekeon 等, Serpent 算法扩散层采用移位、循环移位和异或构成, 扩散层的 S 盒分支数是 3. Noekeon 采用 Lai-Massey 结构构造扩散层. 还有一些其他的扩散层构造方式, 如 PRIDE [7], 结合比特置换和 MDS 码, 采用了计算机搜索的方式构造便于在微处理器上高速实现.

我们致力于设计一个安全性强, 软硬件运行效率高, 结构简单, 易于实现, 灵活性强的分组密码算法, 不仅适用于计算机通信网络, 还可以用于物联网、移动通信网络等, 提供安全高效的密码保护.

本文提出的 FESH 算法基于国际上分组密码设计广泛采用的 SPN 结构, 通过迭代混淆层 S 盒和扩散层字混合提供强雪崩效用, 不仅保证强的安全性, 还兼顾了软硬件实现. 算法支持 128 比特分组和 256 比特分组, 两种不同的分组长度采用相同的轮函数结构实现, 方案简洁优美. 混淆层采用 4 比特的 S 盒实现, 在 S 盒实现的时候不仅考虑了其安全性, 还考虑 S 盒的软硬件实现情况, 可以转换为字运算实现, 与扩散层的字混合运算结合提供高的实现性能. 扩散层的设计, 我们结合了计算机搜索寻找可以实现强扩散有便于软硬件实现的方案. 我们对算法进行了自评估, 重点针对差分类和线性类分析, 给出了差分分析、回轮攻击、线性分析、不可能差分分析、积分攻击和相关密钥分析. 在分析的过程中, 我们结合了最新的一

些分析方法以及基于 MILP 的自动化搜索等. 我们的算法可以抵抗现有的分析方法, 并且具有较大的安全冗余; FESH 算法软硬件实现效率高, 在 PC 机、ARM 平台和硬件 ASIC 平台下都具有出色的实现性能. 算法 S 盒可以转换为基本逻辑运算, 抗侧信道攻击代价低.

2 FESH 算法描述

FESH 算法分组长度为 n 比特, 密钥长度为 m 比特, 记为 FESH-n-m, 本算法包括 FESH-128-128, FESH-128-192, FESH-128-256, FESH-256-256, FESH-256-384, FESH-256-512 六个版本, 每个版本的轮数 N, 见表 1. FESH 算法采用 SPN 结构, 包括加密算法、解密算法、密钥调度算法三部分.

表 1 算法版本及轮数

Table 1 Parameters of FESH family

算法版本	分组长度	密钥长度	轮数 N
FESH-128-128	128	128	16
FESH-128-192	128	192	20
FESH-128-256	128	256	20
FESH-256-256	256	256	24
FESH-256-384	256	384	28
FESH-256-512	256	512	28

FESH 算法的加解密基于字 (word) 操作, 比特串到字的生成规则如下: 每 8 个连续的比特串 $b_0b_1b_2b_3b_4b_5b_6b_7$ 组成一个字节, 其中 b_0 为最高有效位, b_7 为最低有效位; 从字节到字的转换采用大端序, 即将低位字节存放在内存的高地址, 高位字节存放在内存的低地址.

128 比特分组可以表示为 4 个 32 比特字 (w_0, w_1, w_2, w_3), 每一行代表一个字, 如图 1. 256 比特分组可以表示为 4 个 64 比特字, 每一行代表一个 64 比特的字.

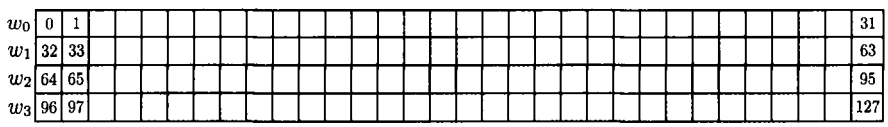


图 1 状态与字的转换

Figure 1 State and words

2.1 符号说明

为了便于算法描述, 首先列出本文中用到的主要的符号.

P:	明文	$\sim x$:	x 按比特取反
C:	密文	$x \bigoplus y$:	x 和 y 进行按位异或运算
$x \parallel y$:	字符串 x 和 y 的级联	n:	分组长度, 取决于算法版本
N:	算法轮数, 大小依赖算法版本	m:	密钥长度
$x \lll s$:	字 x 循环左移 s 比特	RK_i:	轮密钥, 长度为 n 比特
$x \bigvee y$:	x 和 y 进行按位或运算	$K[i \sim j]$:	从左开始取密钥 K 的第 i 到第 j 比特
$x \bigwedge y$:	x 和 y 进行按位与运算		

2.2 加密算法

FESH 算法基于 SPN 结构, 首先将明文分组 P 与白化密钥 (RK_0) 异或, 然后进行 N 轮 (轮函数) 迭代运算, 轮函数包括: S 盒替换 (SubNibble)、字混合 (MixWord) 和轮密钥加操作, 见图 2. 其中 $\mathrm{RK}_1, \mathrm{RK}_2, \cdots, \mathrm{RK}_N$ 为轮密钥, 由密钥调度算法生成. 最后输出为密文 C. 加密算法伪代码如算法 1.

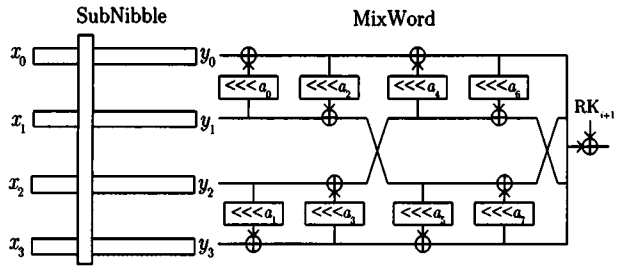


图 2 FESH 算法轮函数示意图

Figure 2 Round function of FESH

算法 1 加密算法

$X_0 = P \oplus \mathrm{RK}_0$
for $i = 0 \to N-1$ **do**
  $Y_i = \mathrm{SubNibble}\ (X_i)$
  $Z_i = \mathrm{MixWord}\ (Y_i)$
  $X_{i+1} = Z_i \oplus \mathrm{RK}_{i+1}$
end
$C = X_N$

S 盒替换 (SubNibble)

S 盒替换是基于半字节的非线性替换, 其将 n 比特状态 X 划分为四个 $n/4$ 比特的字 (x_0, x_1, x_2, x_3), 取出每个字的第 i 比特, 合并为半字节 $s_i = (x_{0i} \parallel x_{1i} \parallel x_{2i} \parallel x_{3i})$, 利用该半字节查询 S 盒, 用获得的新值替换原先的值, 获得输出状态 Y, Y= SubNibble(X). S 盒替换表见表 2, 例如, S 盒输入为 1, 对应的 S 盒的输出为 13. S 盒替换表 S0 的指令实现见附录 A.

表 2 S 替换表 S0

Table 2 Substitution table of SubNibble: S0

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

字混合 (MixWord)

字混合运算是基于字的线性运算, 对于 S 盒替换后的状态 Y 划分为四个字 (y_0, y_1, y_2, y_3), 然后执行下面的运算, 获得 $Z = (z_0, z_1, z_2, z_3)$= MixWord(Y), 见图 3.

$$
\begin{cases}
y_0 = y_0 \oplus (y_1 <<< a_0) \\
y_3 = y_3 \oplus (y_2 <<< a_1) \\
y_1 = y_1 \oplus (y_0 <<< a_2) \\
y_2 = y_2 \oplus (y_3 <<< a_3) \\
t = y_1 \\
y_1 = y_2 \\
y_2 = t
\end{cases}
\qquad
\begin{cases}
y_0 = y_0 \oplus (y_1 <<< a_4) \\
y_3 = y_3 \oplus (y_2 <<< a_5) \\
y_1 = y_1 \oplus (y_0 <<< a_6) \\
y_2 = y_2 \oplus (y_3 <<< a_7) \\
z_0 = y_0 \\
z_1 = y_2 \\
z_2 = y_1 \\
z_3 = y_3
\end{cases}
$$

对于分组长度 128 比特的分组, 字长为 32 比特, 对于分组长度为 256 比特的分组, 字长为 64 比特, 两种分组长度的字混合运算的移位常数 $a_0, \cdots, a_7$ 见表 3.

轮密钥加

加密时, 首先将明文分组逐比特异或白化密钥 RK_0. 在第 r 轮 $(r = 1, 2, \cdots, N)$ 的加密过程中, 将字混合运算后状态 Z 逐比特异或轮密钥 RK_r, 获得输出状态.

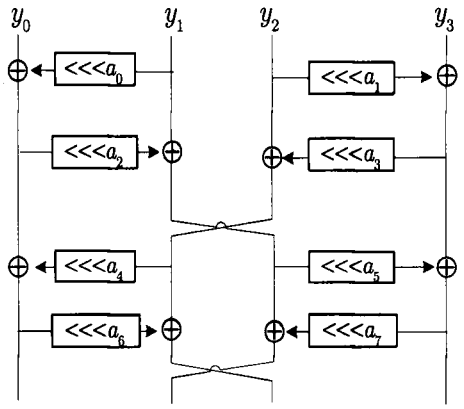


图 3 FESH 算法的字混合示意图
Figure 3 MixWord of FESH

表 3 字混合参数
Table 3 Parameters of MixWord

版本	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
FESH-128-128/192/256	29	13	4	21	15	19	25	6
FESH-256-256/384/512	58	33	8	1	17	44	5	9

2.3 解密算法

解密算法为加密算法的逆运算. 进行 N 轮 (轮函数) 迭代运算, 迭代运算包括: 解密轮密钥加操作、字混合逆运算 (MixWord^{-1})、S 盒替换逆运算 (SubNibble^{-1}). 最后输出与白化密钥 RK_0 异或获得明文, 解密算法伪代码如算法 2.

算法 2 解密算法

$X_N = C$
for $i = N-1 \to 0$ **do**
  $Z_i = X_{i+1} \oplus \text{RK}_{i+1}$
  $Y_i = \text{MixWord}^{-1}(Z_i)$
  $Xi = \text{SubNibble}^{-1}(Y_i)$
end
$P = X_0 \oplus \text{RK}_0$

解密轮密钥加

解密时, 初始状态为密文, 在第 r 轮 ($r = 1, 2, \cdots, N$) 的解密过程中, 将状态逐比特异或轮密钥 RK_{N-r+1}, 将最后一个状态异或白化密钥 RK_0 获得输出明文.

字混合逆运算 (MixWord^{-1})

将上一步获得的状态 Z 划分为四个字 (z_0, z_1, z_2, z_3), 然后执行下面的运算, 获得 $Y = (y_0, y_1, y_2, y_3) = \text{MixWord}^{-1}(Z)$. 分组长度 128 比特和分组长度 256 比特的字混合逆运算的移位常数 $a_0, \cdots, a_7$ 与字混合运算所采用的移位常数相同, 见表 3.

S 盒替换逆运算 (SubNibble^{-1})

S 盒替换逆运算是基于半字节的非线性替换, 其将 n 比特状态 Y 划分为四个字 (y_0, y_1, y_2, y_3), 取出每个字的第 i 比特, 合并为半字节 $s_i = (y_{0i}||y_{1i}||y_{2i}||y_{3i})$, 利用该半字节查询 S 盒逆向替换表, 用获得的新值替换原先的值, 获得输出状态 X, $X = \text{SubNibble}^{-1}(Y)$. S 盒逆运算见表 4, 例如, 输入为 7, 对应的输出为 5.

2.4 密钥调度

密钥调度算法采用 F 函数生成轮密钥 RK_i. F 函数采用简化的轮函数生成轮密钥.

- 对于算法 FESH-128-128 和 FESH-256-256, 密钥长度与分组长度相等, 其密钥调度算法利用函数

表 4 S 盒逆向替换表

Table 4 Substitution table of SubNibble^{-1}

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

F 和密钥 K 生成轮密钥 $\text{RK}_i(i=0,1,\cdots,N)$.

$$\begin{aligned}&\text{RK}_0 = K\\&\text{RK}_i = F(\text{RK}_{i-1}, \text{Cst}_{i-1}), i=1,\cdots,N\end{aligned}$$

- 对于算法 FESH-128-256 和 FESH-256-512, 密钥长度为分组长度的两倍, 密钥调度算法首先将 $2n$ 长度的主密钥 K 分割为两个 n 比特的轮密钥 RK_0 和 RK_1, 然后采用 F 函数生成其他轮密钥, 见图4.

$$\begin{aligned}&\text{RK}_0 = K[0\sim n-1]\\&\text{RK}_1 = K[n\sim 2n-1]\\&\text{RK}_{i+1} = F(\text{RK}_i, \text{Cst}_{i-1})\oplus \text{RK}_{i-1}, i=1,\cdots,N-1\end{aligned}$$

- 其他密钥长度时, $n<|K|<2n$, 首先将密钥填充为 $2n$ 长度. 填充方案为直接在左边补 0 填充成 $2n$ 比特. 如果密钥长度是 $1.5n$ 比特时, 我们在密钥前面填充 $0.5n$ 个 0 使之为 $2n$ 比特. 然后采用 $2n$ 比特长度的密钥调度方法, 区别是 F 函数采用的常数是不同的, 本算法给出了 FESH-128-192 和 FESH-256-384 的两种情况的密钥调度如下:

$$\begin{aligned}&\text{RK}_0 = 0\cdots 0||K[0\sim 0.5n-1]\\&\text{RK}_1 = K[0.5n\sim 1.5n-1]\\&\text{RK}_{i+1} = F(\text{RK}_i, \text{Cst}_{i-1})\oplus \text{RK}_{i-1}, i=1,\cdots,N-1\end{aligned}$$

F 函数是简化的轮函数, 包括常量加 (ARC)、查 S 盒 (SubNibble_K) 和密钥字混合 (MixWord_K) 运算, 见图5. 轮函数 $Z_K = F(\text{RK}_i, \text{Cst}_{i-1})$ 的伪代码如下:

$$\begin{aligned}&X_K = \text{ARC}(\text{RK}_i, \text{Cst}_{i-1}) = \text{RK}_i(\text{Cst}_{i-1}, 0, 0, 0)\\&Y_K = \text{SubNibble}_K(X_K)\\&Z_K = \text{MixWord}_K(Y_K)\end{aligned}$$

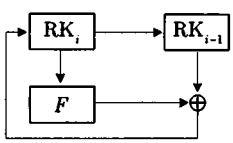


图 4 密钥生成示意图

Figure 4 Round key generation

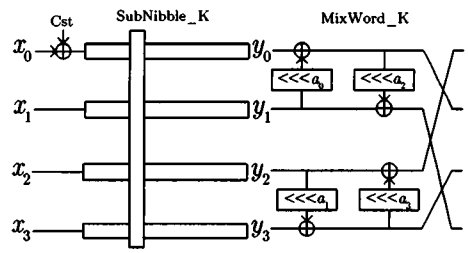


图 5 F 函数示意图

Figure 5 Function F

常量加 ARC 通过异或常数, 破坏密钥调度算法的对称特性, 每一轮的常数都不同. 初始值 Cst_0 是一个常数字, 分组长度 128 时 Cst_0 是 32 位, 分组长度 256 时 Cst_0 是 64 位. 不同的算法版本采用不同的常数初始值, 见表5, 其余常数由初始常数通过循环移位生成, 用于密钥调度过程中, 第 i 个常数 $\text{Cst}_i = \text{Cst}_0 \lll i$. 常数从圆周率的小数部分选取.

SubNibble_K 采用 4 比特的 S 盒, 也是基于半字节的非线性替换, 其将 n 比特状态 X_K 划分为四个

表 5 初始常量表
Table 5 Table of initial constant

版本	常量 $\mathrm{Cst_0}$
FESH-128-128	0x243F6A88
FESH-128-192	0x85A308D3
FESH-128-256	0x13198A2E
FESH-256-256	0x03707344A4093822
FESH-256-384	0x299F31D0082EFA98
FESH-256-512	0xEC4E6C89452821E6

字 (x_0, x_1, x_2, x_3), 取出每个字的第 i 比特, 合并为半字节 $s_i = (x_{0i} \parallel x_{1i} \parallel x_{2i} \parallel x_{3i})$, 利用该半字节查询 S 盒, 用获得的新值替换原先的值, 获得输出状态 Y_K, $Y_K = \mathrm{SubNibble}_K(X_K)$. 采用 4 比特的 S 盒见表6. S 盒替换表 S1 的指令实现见附录 A.

表 6 密钥调度采用的 S 盒替换表 S1
Table 6 Substitution table of $\mathrm{SubNibble}_K$: S1

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

$\mathrm{MixWord}_K$ 基于字的线性运算, 密钥调度中字混合运算采用轮函数字混合运算的简化版, 见图6, 输入 $Y_K = (y_0, y_1, y_2, y_3)$, 输出 $Z_K = (z_0, z_1, z_2, z_3) = \mathrm{MixWord}_K(Y_K)$, 运算如下:

$$\begin{cases} y_0 = y_0 \oplus (y_1 <<< a_0) \\ y_3 = y_3 \oplus (y_2 <<< a_1) \\ y_1 = y_1 \oplus (y_0 <<< a_2) \\ y_2 = y_2 \oplus (y_3 <<< a_3) \\ z_0 = y_2 \\ z_1 = y_0 \\ z_2 = y_3 \\ z_3 = y_1 \end{cases}$$

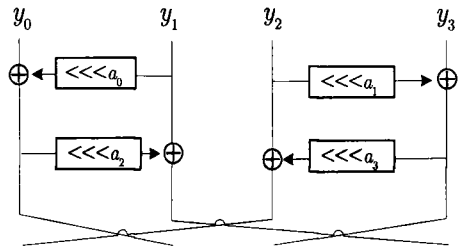


图 6 密钥调度中的字混合示意图
Figure 6 $\mathrm{MixWord}_K$ in key schedule

分组长度为 128 和 256 的版本密钥调度中字混合运算 ($\mathrm{MixWord}_K$) 采用的参数 a_0, a_1, a_2, a_3 见表7.

3 算法的设计原理

我们致力于设计一个安全性强, 软硬件运行效率高, 结构简单, 易于实现, 灵活性强的分组密码算法, 不仅适用于计算机通信网络, 还可以便于移植物联网、移动通信网络等, 提供安全高效的密码保护. 算法

表 7 密钥调度中字混合参数

Table 7 Parameters of MixWord_K in key schedule

版本	a_0	a_1	a_2	a_3
FESH-128-128/192/256	24	30	7	18
FESH-256-256/384/512	1	18	50	24

支持 128 比特的分组和 256 比特的分组, 采用统一的轮函数结构, 设计方案简洁. 在保证算法安全的条件下, 要兼顾软件实现效率, 硬件实现效率和硬件资源, 同时算法要具有一定的灵活性, 在 8 位、32 位和 64 位等不同的软件处理平台和 FPGA、ASIC 硬件平台上都具有较好的性能.

S 盒设计

本算法采用了 4 比特的 S 盒, S 盒的构造不仅考虑了常规的安全特征, 如差分概率、线性偏差优势、代数次数、固定点等, 还特别考虑了低汉明重量的差分分布情况, 以及低汉明重量的线性掩码分布. Saarinen 等 [9] 对所有 4-4 的活性 S 盒进行分析, 证明 "最优" 的 S 盒共 16 个等价类, 我们对其给出的最大 BN = 3 的类进一步分析, 寻找输入输出 1 比特的线性掩码数量最小的, 最小值为 4, 最后发现只有两个等价类中存在这种情况, 基于此我们从这两个等价类构造 S 盒, 从中寻找 bit-slice 指令运行效率高的 S 盒, 提高算法实现性能. 张文涛等在 RECTANGLE 算法的设计中也考虑了低汉明重量差分和线性的情况, 采用 BN = 2 的类 [8]. S 盒 S0 的指标如下:

- S 盒是置换, 是平衡的, 没有固定点;
- 最大差分概率为 2^{-2}, 输入输出是 1 比特的差分个数为 0;
- 最大线性逼近概率 2^{-2}, 输入输出掩码是 1 比特的线性偏差个数是 4;
- S 盒代数次数最高为 3, S 盒逆的代数次数最大为 3;
- S 盒便于采用 bit-slice 实现, 指令数为 15, 具有较高的软件实现效率.

扩散层设计

在算法扩散层构造方面, 我们希望扩散层便于求逆, 逆向运算与正向运算性能相当, 同时也考虑扩散层与 S 盒的运算资源及实现效率, 产生强的扩散, 要求每个比特采用的异或个数不能超过 2 个. 考虑到可逆性, 我们利用 2 分支、4 分支的 Feistel 结构和 MISTY 结构构造扩散层, 通过计算机搜索比较其扩散性能及实现效率, 最后采用 8 个字循环移位和 8 个字异或构成的深度为 4 的 4 分支 Feistel 结构作为扩散层. 该扩散层分支数为 5, 参数选取使分支数达到最优, 在与 S 盒搭配上更具优势. 线性层打破 S 盒的结构, 增加了算法的扩散强度. 字混合运算性质如下:

- 扩散层由四分支 Feistel 结构构成, 包括 8 个循环移位、8 个异或, 异或深度为 4.
- 扩散层分支数为 5, 输入 2 个活性盒时, 输出至少为 3 个, 或相反.

轮函数设计中还综合考虑了扩散层与 S 盒的结合, 特别是针对线性偏差汉明重量 1 比特的情况进行考虑, 128 比特分组长度保证 3 轮线性逼近路线活性盒最少为 14, 3 轮差分路线活性盒最少为 16.

密钥调度算法

密钥调度算法具有高效的软硬件实现效率, 比轮函数简单. 密钥调度算法是置换, 不存在等价密钥, 与算法搭配抗相关密钥、弱密钥攻击等. 密钥调度算法采用简化的轮函数 F, 是可逆的, 采用更高运行效率的 S 盒 S1, S1 最大差分概率为 2^{-2}, 最大线性逼近概率 2^{-2}, S1 及其逆的最高代数次数均为 3, 无固定点, S1 软件实现仅需要 10 条指令. 密钥扩展算法采用深度为 2 的四分支 Feistel 结构的字混合运算进行扩散, 密钥扩散与轮函数扩散尽可能不相交, 能够有效抵抗相关密钥攻击、弱密钥攻击等.

4 算法的安全性分析

我们对分组密码 FESH 的安全性进行了自评估, 主要考虑了差分分析、回轮攻击、线性分析、不可能差分分析、积分攻击和相关密钥分析等, 对应上述分析方法, 我们将 FESH-128 的区分器和分析轮数汇总

在表8, 将 FESH-256 的区分器和分析轮数汇总在表9.

表 8 FESH-128 的分析结果汇总
Table 8 Cryptanalysis of FESH-128

攻击类型	FESH-128-128		FESH-128-256	
	区分器轮数	分析轮数	区分器轮数	分析轮数
差分分析	5 轮	6 轮	5 轮	7 轮
回轮攻击	6 轮	7 轮	6 轮	8 轮
线性分析	5 轮	6 轮	5 轮	7 轮
不可能差分	3.5 轮	6 轮	3.5 轮	7 轮
积分攻击	7 轮	9 轮	7 轮	10 轮
相关密钥回轮攻击	7 轮	9 轮	9 轮	13 轮

表 9 FESH-256 的分析结果汇总
Table 9 Cryptanalysis of FESH-256

攻击类型	FESH-256-256		FESH-256-512	
	区分器轮数	分析轮数	区分器轮数	分析轮数
差分分析	5 轮	6 轮	5 轮	7 轮
回轮攻击	7 轮	9 轮	7 轮	11 轮
线性分析	6 轮	8 轮	6 轮	9 轮
不可能差分	4.5 轮	7 轮	4.5 轮	8 轮
积分攻击	8 轮	10 轮	8 轮	11 轮
相关密钥攻击	7 轮	9 轮	9 轮	13 轮

差分类分析

对于 FESH-128, 我们通过搜索和评估, 证明了 FESH-128 的 3 轮差分路线最小活跃盒数为 16, 4 轮差分路线中活跃 S 盒个数至少不小于 25. 根据 4 轮和 3 轮最小活跃 S 盒个数的估计, 我们证明 $(4+4+3=)11$ 轮 FESH-128 差分最小活跃盒个数至少为 $(25+25+16=)66$, 从而证明了不存在有效的 11 轮 FESH-128 差分路线. 进一步扩展搜索, 获得 5 轮差分路线, 共 48 个活性 S 盒, 在该路线的尾部扩展一轮攻击 6 轮. 利用 3 轮和 2 轮的差分路线构造回轮 (Boomerang) 区分器, 可构造 $3+1+2=6$ 轮区分器, 向后扩展 1 轮, 给出 7 轮 FESH-128-128 的分析. 利用相同的路线, 可以攻击 8 轮 FESH-128-256.

对于 FESH-256, 我们搜索计算得出 4 轮差分最小活跃盒个数至少为 32, 因此 16 轮 FESH-256 差分最小活跃盒个数至少为 128 个, 从而证明了不存在有效的 16 轮 FESH-256 差分路线. 给出了 5 轮差分路线, 共 79 个活性 S 盒, 可以分析 6 轮的 FESH-256-256 和 7 轮的 FESH-256-512. 3 轮路线活性盒为 $(5 \to 1 \to 12)$ 概率小于等于 2^{-36}. 利用两条 3 轮的差分路线构造回轮区分器, 可构造 $3+1+3=7$ 轮区分器, 前后各扩展 1 轮, 最多给出 9 轮 FESH-256-256 的分析; 对于 FESH-256-512, 采用相同的路线, 在 9 轮的分析结果上至多增加 2 轮, 给出 11 轮的分析结果.

线性分析

通过搜索, 可证明 FESH-128 的 3 轮线性路线最小活跃 S 盒个数为 14. 对 4 轮线性路线进行与 4 轮差分路线类似的分析, 计算得出 4 轮线性路线中活跃 S 盒个数至少不小于 25. 根据四轮和三轮最小活跃 S 盒个数的估计, 我们证明 $(4+4+3=)11$ 轮 FESH-128 线性最小活跃盒个数至少为 $(25+25+14=)64$, 从而证明了不存在有效的 11 轮 FESH-128 线性路线. 给出 5 轮线性路线, 共 46 个活性 S 盒, 可攻击 6 轮 FESH-128-128 或 7 轮 FESH-128-256.

类似于 FESH-128, 我们计算出 4 轮 FESH-256 线性最小活跃盒个数至少为 32, 因此 16 轮 FESH-256 线性最小活跃盒个数至少为 128 个. 给出 6 轮线性路线, 共 99 个活性 S 盒. 将该 6 轮线性路线尾部添加

一轮, 最后一轮共有 56 个线性活跃盒. 将该 6 轮线性路线头部添加一轮, 可以分析 7 轮的 FESH-256-256 和 8 轮的 FESH-256-512.

不可能差分分析

我们采用 Sasaki 等 [11] 和 Cui [12] 等提出的基于 MILP 的不可能差分路线搜索方法, 找到了 FESH-128 的 3.5 轮不可能差分路线, 即头尾各一个 S 盒为活跃盒, 是不可能差分路线. 向上向下各扩展一轮, 各产生 18 个活跃盒, 尾部的线性层对分析影响不大, 可以转换为等价密钥. 基于该路线, 可以分析 6 轮 FESH-128-128 和 7 轮 FESH-128-256. 找到了 FESH-256 的 4.5 轮不可能差分路线, 即头尾各有一个活跃盒, 是不可能差分路线. 向上向下各扩展一轮, 各产生 18 个活跃盒. 可以攻击 7 轮 FESH-256-256 和 8 轮 FESH-256-512.

积分攻击

我们利用 Xiang 等 [13] 提出的基于 MILP 自动化搜索积分路线的方法, 找到了 7 轮 FESH-128 的积分路线, 输入 124 个活跃比特, 输出两个平衡比特. 利用该 7 轮积分路线, 向下扩展 2 轮可以攻击 9 轮 FESH-128-128, 也可以用于攻击 10 轮 FESH-128-256.

利用相同的方法, 我们得到了 8 轮 FESH-256 的积分区分器, 输入 252 个活跃比特, 输出两个平衡比特. 利用该 8 轮积分路线, 向下扩展 2 轮可以攻击 10 轮 FESH-256-256; 对于 FESH-256-512, 路线一样, 可多分析一轮.

相关密钥攻击

主要考虑 FESH-128-128 的抗相关密钥差分分析. 找到了 3 轮相关密钥差分路线. 路线包含 5 个活跃 S 盒: $0 \to 3 \to 2$, 概率等于 2^{-14}; 4 轮相关密钥差分路线, 包含 23 个活跃 S 盒: $0 \to 3 \to 2 \to 18$, 概率等于 2^{-51}. 利用 3 轮相关密钥差分路线, 可以利用 3 轮加 1 轮加 3 轮构造 7 轮回轮区分器, 上下添加一轮能攻击 9 轮 FESH-128-128. 对于 FESH-128-256, 由于密钥量是 FESH-128-128 两倍, 考虑密钥关系, 可以将 FESH-128-128 的三轮相关密钥差分路线扩展到 4 轮. 利用 4 轮加 1 轮加 4 轮, 构造 9 轮回轮区分器, 密钥恢复时, 最多上下各增加 2 轮, 可以攻击至多 13 轮 FESH-128-256.

找到了 FESH-256-256 的三轮相关密钥差分路线, 概率为 2^{-12}. 可以用该路线构造 3 轮加 1 轮加 3 轮的回轮区分器, 并攻击 9 轮 FESH-256-256. 给出了 4 轮 FESH-256-256 相关密钥差分路线, 概率为 2^{-116}. 对于 FESH-256-512, 由于密钥量是 FESH-256-256 两倍, 考虑密钥关系, 可以将 FESH-256-256 的三轮相关密钥差分路线, 扩展到 4 轮. 利用 4 轮加 1 轮加 4 轮, 构造 9 轮回轮区分器, 最多上下各添加 2 轮, 可以攻击 13 轮 FESH-256-512.

这些分析方法涵盖了目前主流的密码分析方法, 因此, 在考虑现有分析方法的情况下, FESH 算法各版本的轮数具有一定的冗余. FESH-128-128 最多可以分析到 9 轮, 占总轮数 16 的 56%, FESH-128-256 可以分析到 13 轮, 占总轮数 20 的 65%, 具有较大的安全冗余. FESH-256-256 最多可以分析到 10 轮, 占总轮数 24 的 42%, FESH-256-512 可以分析到 13 轮, 占总轮数 28 的 46%, 也具有很大的安全冗余.

5 算法的软硬件实现

5.1 软件实现及性能

FESH 算法便于软件实现, 算法可以采用基于字的基本逻辑运算实现, 不需要任何查表操作, 基本的 C 语言实现具有较高的性能, 同时在 Intel CPU 架构下可以很方便的结合扩展指令集 SSE 和 AVX 实现分组密码的并行处理. 我们采用标准 C 实现了 FESH 算法, 分别在 PC 机和 ARM 的环境下对算法进行测试. PC 机测试平台为: CPU Intel Core i7-4790QM CPU @ 3.60 GHz, 内存 8.0 GB, 操作系统 Windows 7 专业版 64 位, 编译环境 Microsoft Visual Studio 2015. 算法性能如表 10.

我们也评估了 FESH 算法在 ARM 平台下的实现性能, 选用的 ARM32 平台为基于 STM32 F103 的开发板 (主频 72 Mhz): stm32f1zet6 核心板 6, 512 K 片内 Flash, 64 K 片内 RAM. 性能测试采用 CBC 模式加密 256 B 运行 100 000 次, 求平均值. 在 ARM32 平台下实现性能见表 11.

表 10 FESH 算法软件速度 (Mbps)
Table 10 Performance of FESH in software

算法	实现方式	算法版本	加密 256 B	解密 256 B
FESH	单路实现	FESH-128-128	1429	1240
		FESH-128-256	1134	1004
		FESH-256-256	1581	1413
	8 路实现	FESH-128-128	4418	4418
		FESH-128-256	3699	3664
		FESH-256-256	3212	3191

表 11 FESH 算法 ARM 平台下软件速度 (Mbps)
Table 11 Performance of FESH in ARM

算法版本	加密 256 B	解密 256 B
FESH-128-128	8.60	8.34
FESH-128-256	7.00	6.97
FESH-256-256	4.18	4.30

5.2 硬件实现及性能

我们用 Verilog 采用 on-the-fly 模式对 FESH 算法进行实现, 加解密采用同一个模块实现, 测试加解密性能, 还单独对加密算法进行了测试. 算法轮数为 N, 在实现中增加了 1 个周期控制信号, 因此算法运行周期按照 $N+1$ 计算. 我们利用 Sysnopsys Design Compiler, ASIC 平台工艺库为 SMIC 40 nm 工艺库, 对算法的三个版本进行了综合, 加解密性能见表 12. 我们也对算法的加密模块进行综合, 实现性能见表 13. 加密 32 个分组, 解密速度是加密速度的 32/33, 解密比加密多了一次密钥调度.

表 12 FESH 算法加解密在 ASIC 平台实现性能
Table 12 Performance of FESH in ASIC

对比项	FESH-128-128	FESH-128-256	FESH-256-256
运算周期	17	21	25
时钟约束 (ns)	0.46	0.48	0.47
加密速率 (Mbps)	16368	12698	21787
解密速率 (Mbps)	15 872	12 313	21 126
面积 (um^2)	15957	17137	28097

表 13 FESH-128-128 算法加密在 ASIC 平台实现性能
Table 13 Performance of FESH-128-128 in ASIC

算法	加密周期	综合方式 (时钟约束)	频率 (MHz)	面积 (gate)	速度 (Mbps)	吞面比 (Kbps/gate)
FESH-128-128 40 nm	17	0.44	2272	8869	17112	1929.44
		0.66	1515	4781	11408	2385.78
FESH-128-128 130 nm	17	1.25	800	7834	6023	768.84
		2.5	400	4531	3011	664.6
AES-128-128 [14] 130 nm	11		145.35	12454	1691.35	135.81
	54		131.24	5398	311.09	57.63

6 结论

本算法采用分组密码设计广泛采用的替换-置换网络 (SPN) 结构构建, 构造方案简洁便于实现, 算法兼顾了安全性和软硬件实现性能. 轮函数采用了比特切片模式, 与 AES 竞赛中的 Serpent 的轮函数相似, 但是采用的 S 盒安全强度更高优于 Serpent 的 S 盒, 轮函数中的 S 盒不存在输入输出汉明重量都是 1 的差分, 而汉明重量为 1 的输入和输出的线性掩码仅有 4 个. 而且 S 盒给出的比特切片实现指令只有 15 条, 也是少于 Serpent 的 S 盒的实现指令数. 我们采用的字混合运算基于 4 分支 Feistel 结构变种, 各分支扩散比较均衡, 而且活性盒最小分支数为 5, 大于 Serpent 的最小分支数. 因此我们的轮函数比 Serpent 具有更强的抗差分和线性攻击的能力, 目前我们发现分组长度 128 的 5 轮的差分路线和 5 轮的线性路线概率都比较接近于临界值. FESH 算法对 128 比特的分组和 256 比特的分组, 采用统一的结构, 都可以转化为基于字的基本逻辑运算, 这种实现方式在现代 Intel CPU 架构下可以很方便的结合扩展指令 SSE 和 AVX 实现分组密码的并行处理, 支持 ECB、CTR 和 XTS 等模式下分组密码的高效实现, 同时算法采用基于字的逻辑运算代替查表运算, 在抗侧信道攻击具有优势. 本算法共 6 个版本便于软硬件实现, 支持多种密钥长度包括 128 比特、192 比特、256 比特、384 比特和 512 比特, 包含了国际上主流的分组长度和密钥长度, 可以应用于各种安全协议中, 具有兼容性.

References

[1] DAEMEN J, RIJMEN V. Specification of Rijndael[M]. In: The Design of Rijndael: AES—The Advanced Encryption Standard. Springer Berlin Heidelberg, 2013: 31–51. [DOI: 10.1007/978-3-662-04722-4_3]

[2] BANIK S, BOGDANOV A, ISOBE T, et al. Midori: A block cipher for low energy[C]. In: Advances in Cryptology—ASIACRYPT 2015. Springer Berlin Heidelberg, 2015: 411–436. [DOI: 10.1007/978-3-662-48800-3_17]

[3] BEIERLE C, JEAN J, KÖLBL S, et al. The SKINNY family of block ciphers and its low-latency variant MANTIS[C]. In: Advances in Cryptology—CRYPTO 2016, Part II. Springer Berlin Heidelberg, 2016: 123–153. [DOI: 10.1007/978-3-662-53008-5_5]

[4] BILGIN B, NIKOVA S, NIKOV V, et al. Threshold implementations of all 3 × 3 and 4 × 4 S-boxes[C]. In: Cryptographic Hardware and Embedded Systems—CHES 2012. Springer Berlin Heidelberg, 2012: 76–91. [DOI: 10.1007/978-3-642-33027-8_5]

[5] BIHAM E, ANDERSON R, KNUDSEN L. Serpent: A new block cipher proposal[C]. In: Fast Software Encryption—FSE 1998. Springer Berlin Heidelberg, 1998: 222–238. [DOI: 10.1007/3-540-69710-1_15]

[6] DAEMEN J, PEETERS M, VAN ASSCHE G, et al. Nessie proposal: NOEKEON[C]. In: Proceedings of First Open NESSIE Workshop. Leuven, Belgium. 2000: 213–230.

[7] ALBRECHT M R, DRIESSEN B, KAVUN E B, et al. Block ciphers—Focus on the linear layer (feat. PRIDE). In: Advances in Cryptology—CRYPTO 2014, Part I. Springer Berlin Heidelberg, 2014: 57–76. [DOI: 10.1007/978-3-662-44371-2_4]

[8] ZHANG W, BAO Z, LIN D, et al. RECTANGLE: A bit-slice lightweight block cipher suitable for multiple platforms[J]. SCIENCE CHINA Information Sciences, 2015, 58(12): 1–15. [DOI: 10.1007/s11432-015-5459-7]

[9] SAARINEN M J O. Cryptographic analysis of all 4 × 4-bit S-boxes[C]. In: Selected Areas in Cryptography—SAC 2011. Springer Berlin Heidelberg, 2011: 118–133. [DOI: 10.1007/978-3-642-28496-0_7]

[10] BOGDANOV A, KNUDSEN L R, LEANDER G, et al. PRESENT: An ultra-lightweight block cipher[C]. In: Cryptographic Hardware and Embedded Systems—CHES 2007. Springer Berlin Heidelberg, 2007: 450–466. [DOI: 10.1007/978-3-540-74735-2_31]

[11] SASAKI Y, TODO Y. New impossible differential search tool from design and cryptanalysis aspects—Revealing structural properties of several ciphers[C]. In: Advances in Cryptology—EUROCRYPT 2017, Part III. Springer Cham, 2017: 185–215. [DOI: 10.1007/978-3-319-56617-7_7]

[12] CUI T, JIA K, FU K, et al. New automatic search tool for impossible differentials and zero-correlation linear approximations[J]. IACR Cryptology ePrint Archive, 2016, 2016/689.

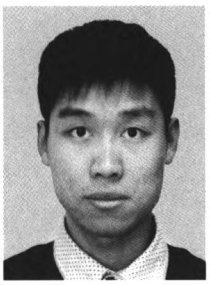
[13] XIANG Z, ZHANG W, BAO Z, et al. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers[C]. In: Advances in Cryptology—ASIACRYPT 2016, Part I. Springer Berlin Heidelberg, 2016: 648–678. [DOI: 10.1007/978-3-662-53887-6_24]

[14] SATOH A, MORIOKA S. Hardware-focused performance comparison for the standard block ciphers AES, Camellia, and Triple-DES[C]. In: Information Security—ISC 2003. Springer Berlin Heidelberg, 2003: 252–266. [DOI:

10.1007/10958513_20]

作者信息

贾珂婷 (1983–), 山东青岛人, 助理研究员. 主要研究领域为对称密码算法的安全性分析与设计.
ktjia@tsinghua.edu.cn

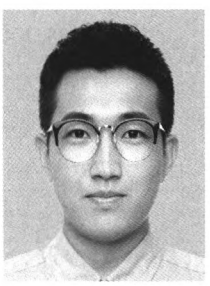


董晓阳 (1988–), 河北肃宁人, 助理研究员. 主要研究领域为对称密码算法的安全性分析与设计.
xiaoyangdong@tsinghua.edu.cn

魏淙洛 (1994–), 河北沧州人, 博士在读. 主要研究领域为对称密码及其软件优化实现.
wcm16@mails.tsinghua.edu.cn

李铮 (1992–), 山东济南人, 助理研究员. 主要研究领域为对称密码算法的安全性分析与设计.
lizhengcn@bjut.edu.cn

周海波 (1992–), 山东青岛人, 博士生在读, 主要研究领域为对称密码算法的安全性分析与设计.
zhouhaibo@mail.sdu.edu.cn



丛天硕 (1994–), 黑龙江哈尔滨人, 在读博士生. 主要研究领域为密码算法的高效硬件实现.
cts17@mails.tsinghua.edu.cn

附录 A S 盒的软件实现指令

为了便于软件实现, S0 可以采用 bit-slice 实现, 4 比特输入为 $(x_{0i} \parallel x_{1i} \parallel x_{2i} \parallel x_{3i})$, 实现如下:

$$
\begin{cases}
r0 = x_{1i} \\
r1 = x_{2i} \\
r2 = x_{0i} \\
r3 = x_{3i} \\
r4 = r0 \vee r2 \\
r1 = r1 \oplus r4 \\
r4 = r3 \vee r1 \\
r2 = r2 \oplus r4 \\
r1 = r1 \oplus r3 \\
r4 = r0 \wedge r2 \\
r3 = r3 \oplus r4 \\
r2 = r2 \oplus r0
\end{cases}
\quad
\begin{cases}
r0 = r0 \oplus r1 \\
r4 = \sim r2 \\
r4 = r4 \vee r3 \\
r1 = r1 \oplus r4 \\
r4 = \sim r0 \\
r4 = r4 \vee r1 \\
r3 = r3 \oplus r4 \\
y_{0i} = r2 \\
y_{1i} = r0 \\
y_{2i} = r1 \\
y_{3i} = r3
\end{cases}
$$

最后获得 S0 的 4 比特输出为 $(y_{0i} \parallel y_{1i} \parallel y_{2i} \parallel y_{3i})$.

为了便于软件实现, S1 可以采用 bit-slice 实现, 输入 $(x_{0i} \parallel x_{1i} \parallel x_{2i} \parallel x_{3i})$, bit-slice 实现如下:

$$\begin{cases} r3 = x_{0i} \\ r0 = x_{1i} \\ r1 = x_{2i} \\ r2 = x_{3i} \\ r1 = \sim r1 \\ r4 = r1 \vee r3 \\ r2 = r2 \oplus r4 \end{cases} \quad \begin{cases} r4 = r0 \vee r1 \\ r3 = r3 \oplus r4 \\ r4 = \sim r0 \\ r4 = r4 \wedge r2 \\ r1 = r1 \oplus r4 \\ r4 = r2 \wedge r3 \\ r0 = r0 \oplus r4 \end{cases}$$

最后获得 S1 的输出为 $(r1||r3||r2||r0)$.

附录 B 算法测试实例

我们列出针对一个分组的运算实例, 用以验证密码算法实现的正确性. 其中, 数据采用 16 进制表示, 见表 14.

表 14 FESH 测试实例
Table 14 Test data of FESH

算法	数据	值
FESH-128-128	明文	01234567 89abcdef fedcba98 76543210
	加密密钥	01234567 89abcdef fedcba98 76543210
	密文	da99343c 616fe47c 796af018 2cd01915
FESH-128-192	明文	01234567 89abcdef fedcba98 76543210
	加密密钥	01234567 89abcdef fedcba98 76543210 01234567 89abcdef
	密文	50194a2f 461ba122 49378091 6c8e0751
FESH-128-256	明文	01234567 89abcdef fedcba98 76543210
	加密密钥	01234567 89abcdef fedcba98 76543210 01234567 89abcdef fedcba98 76543210
	密文	6da0b2c3 17e99825 ea39b7ca 63d36efd
FESH-256-256	明文	0123456789abcdef fedcba9876543210 0123456789abcdef fedcba9876543210
	加密密钥	0123456789abcdef fedcba9876543210 0123456789abcdef fedcba9876543210
	密文	2a2981f04f011848 c73f7b7a04c1ad39 fb2b1ce7e3a0dc87 16c93c6f2a2c5e07
FESH-256-384	明文	0123456789abcdef fedcba9876543210 0123456789abcdef fedcba9876543210
	加密密钥	0123456789abcdef fedcba9876543210 0123456789abcdef fedcba9876543210 0123456789abcdef fedcba9876543210
	密文	247363c5863e3c71 8bee320818122148 1a1b76a7eac39672 67d22852c97c7a1b
FESH-256-512	明文	0123456789abcdef fedcba9876543210 0123456789abcdef fedcba9876543210
	加密密钥	0123456789abcdef fedcba9876543210 0123456789abcdef fedcba9876543210 0123456789abcdef fedcba9876543210 0123456789abcdef fedcba9876543210
	密文	08fb1029daf2e4bd 8a3b02d8d3c58197 ae163b6da7a623b7 14d8db58413085c7