



中国矿业大学
CHINA UNIVERSITY OF MINING AND TECHNOLOGY

本科生毕业设计（论文）

轻量级分组密码算法 LBlock 的实现和差分 分析研究

Implementation and Differential Analysis of
Lightweight Block Cipher Algorithm Lblock

作 者：李艺帆
导 师：张凤荣

中国矿业大学
2021 年 6 月

中国矿业大学

本科生毕业设计（论文）

轻量级分组密码算法 LBlock 的实现和差分分析
研究

Implementation and Differential Analysis of
Lightweight Block Cipher Algorithm Lblock

作	者	李艺帆	学	号	08173024
导	师	张凤荣	职	称	副教授
学	院	计算机科学与技术学院	专	业	信息安全

二〇〇*年*月

毕业设计（论文）原创性声明

本人郑重声明：所呈交的毕业设计（论文），《 》，是本人
在指导教师指导下，在中国矿业大学攻读学位期间，独立完成的研究工作所取得的成果。据我所知，除文中已经标明引用的内容外，本文
体已经发表或撰写过的研究成果。本人完全意识
担。

标题：黑体小2加粗居中，单倍行距，
段前0.5行，段后0行；
内容：楷体小4，行距固定值20磅

作者签名：

年 月 日

中国矿业大学

(论文) 不包含任何其他个人或集体已经发表或撰写的
文) 所涉及的研究工作作出贡献的其他个人和集体, 均已

标题：黑体小 2 加粗居中，单倍行距，
段前 0.5 行，段后 0 行；
内容：楷体小 4，行距固定值 20 磅

年 月 日

中国矿业大学

本人完全了解中国矿业大学有关收集、保留和使用本人所送交的毕业设计（论文）的规定，即：本科生在校攻读学位期间毕业设计的知识产权单位属中国矿业大学。学校有权保留并向国家有关部门或单位送交论文的复印件和电子版，允许论文被查阅和借阅，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。解密后适用本声明。

标题：黑体小2加粗居中，单倍行距，
段前0.5行，段后0行；
内容：楷体小4，行距固定值20磅

☐ 不保密

☐ 保密，保密期（起讫日期：_____）

作者签名:
年 月 日

导师签名：
年 月 日

致谢

感谢XX
XX
XX
XX
XX

.....
.....
...

标题：黑体小2加粗居中，单倍行距，段前0.5行，段后0行；
内容：楷体小4，行距固定值20磅

...
.....
.....

中国矿业大学本科毕业设计（论文）任务书

设计（论文）题目：		
学院		专业
学生姓名		学

标题：黑体小2加粗居中，单倍行距，段前0.5行，段后0.5行；

正文：宋体小4，行距固定值20磅；

1、设计（论文）的主要内容

2、设计（论文）的基本要求

指导教师签字：

中国矿业大学毕业设计（论文）指导教师评阅书

学生姓名		学号	
设计（论文）题目	<div>标题：黑体小2加粗居中，单倍行距，段前0.5行，段后0.5行； 正文：宋体小4，行距固定值20磅；</div>		
<div>指导教师评语（①基础理论及基本技能的掌握；②研究内容的理论依据和技术方法；③取得的主要成果；④研究量；⑤总体评价及建议成绩；⑥存在问题；⑦是否同意答辩等）：</div>			
成绩：		指导教师签字： 年 月 日	

中国矿业大学毕业设计（论文）评阅教师评阅书

学生姓名		学号	
设计（论文）题目	<div>标题：黑体小2加粗居中，单倍行距，段前0.5行，段后0.5行； 正文：宋体小4，行距固定值20磅；</div>		
<p>评阅教师评语（①选题的意义；②基础理论及专业知识解决实际问题的能力；④工作量的大小；⑤论文的规范程度；⑦总体评价及建议成绩；⑧存在问题；⑨是否同意答辩等）：</p> <div></div> <div>成 绩：</div> <div>评阅教师签字： 年 月 日</div>			

中国矿业大学毕业设计（论文）答辩及综合成绩

答 辩 情 况					
提 出 问 题	回				
	正 确	基 正 确	错 误	错 误	回 答
答辩委员会评语及建议成绩:					
成绩:					
答辩委员会主任					
成绩评定:					
成绩组成	指导教师	评阅教师	答辩成绩	其他	总评
成绩比例					
评分					
学院领导签字:					
年 月 日					

标题：黑体小2加粗居中，单倍行距，段前0.5行，段后0.5行；
正文：宋体小4，行距固定值20磅；

可以根据问题数量添加

不足部分可加页

摘 要

Lblock 算法是由吴文玲等人在 2011 年提出的一种轻量级分组密码算法，它的效率跟高，适应性更强，整个算法的结构与其他常见的 DES，等算法一样都使用的是 Feistel 结构，基于 4 比特模块设计。密钥扩展算法采用非线性移位寄存器，利用 S 盒的变换盒循环移位生成轮密钥。

而差分密码分析是对于攻击分组密码很常见的方法，利用 S 盒的非线性转换，可以针对每一个 S 盒的输出差分进行对输入差分对的猜测，从而得到一份每一个 S 盒的差分表。随后我们通过注入故障，即对进入 S 盒的 4 比特数据进行处理后，得到一个非正常的输出，通过和差分表比对来得到一些可能输入的值的集合。通过每次对不同位置注入 4 比特的故障，来推测出当轮使用的密钥，在往上反推一轮，将已知位置的密钥再扩散到不同位置后再次推断。剩余位置即可通过穷举来探测。

该论文有图 15 幅，表 26 个，参考文献 160 篇。

关键词：LBlock 算法；差分分析；分组加密；非线性混淆；

Abstract

Lblock algorithm is a lightweight block cipher algorithm proposed by Wu Wenling and others in 2011. It has higher efficiency and stronger adaptability. The structure of the whole algorithm, like other common DES algorithms, uses Feistel structure and is based on 4-bit module design. The key expansion algorithm uses nonlinear shift register, and uses the transform box of S-box to generate round key.

Differential cryptanalysis is a very common method to attack block ciphers. By using the nonlinear transformation of S-boxes, we can guess the input differential pairs according to the output differential of each S-box, so as to get a differential table of each S-box. Then we get an abnormal output by injecting a fault, that is, after processing the 4-bit data entering the S-box, we get a set of possible input values by comparing with the difference table. By injecting 4-bit faults into different locations each time, we can infer the key used in the current round. We can infer the key again after one round of backward push, and then spread the key of known location to different locations. The remaining position can be measured by exhaustive method.

Keywords: Lblock algorithm; Differential analysis; Block encryption; Nonlinear confusion.

目 录

摘要·····	I
目录·····	3
1 绪论·····	1
1.1 概述·····	1
1.2 国内外发展现状·····	2
1.3 研究内容·····	3
1.4 论文组织结构·····	3
2 LBlock 算法·····	5
2.1 常见算法符号·····	5
2.2 算法过程·····	5
3 差分分析·····	8
3.1 差分分析基本概念·····	8
3.2 S 盒的差分分布表·····	9
4 算法的实现以及攻击·····	21
4.1 LBlock 算法加密演示·····	21
4.2 一轮差分分析·····	23
4.3 多轮差分分析·····	26
4.4 密钥分析·····	28
4.5 核心代码·····	30
5 结论·····	34
参考文献·····	35
翻译部分·····	115

Contents

Abstract.....	I
Contents.....	3
Introduction.....	1
1.1 Introduction	1
1.2 Development status at home and abroad	2
1.3 research contents	3
1.4 Organizational structure of papers	3
2 Lblock algorithm.....	5
2.1 Common algorithm symbols.....	5
2.2 Algorithm process.....	5
3 Differential analysis	8
3.1 Basic concepts of differential analysis	8
3.2 Differential distribution table of S-box	9
4 Implementation and attack of the algorithm	21
4.1 Lblock algorithm encryption demonstration	21
4.2 A round of differential analysis	23
4.3 Multi round difference analysis	26
4.4 Key analysis	28
4.4 Core program	30
5 Conclusions.....	34
References.....	35

1 绪论

1 Introduction

1.1 研究目标与研究意义 (Research objectives and significance)

1.1.1 研究目标

轻量级分组密码由于其计算, 存储资源开销少且能很好的提供加密性成为了人们研究的热点, 该算法是 2011 年由吴文玲等人在 ANCS 会议上提出的, 整个算法由 32 轮迭代轮加密组成, 密文分组长度为 64 位, 种子密钥长度为 80 位, 采用 Feistel 结构和 SPN 结构, 并且它的加解密过程是互逆的, 所以效率非常高。差分分析是目前最常用的密码分析方法之一, 很多密码分析方法都是基于该算法的衍生和变种。在分组密码的差分分析当中, 通常都是针对功能为混淆且是非线性的 S 盒, 在可知输出差分 and 明文中间状态的情况下选取可能的输入差分, 经过多次的反复的实验探测密钥信息。

查阅文献资料来熟悉整个 LBlock 算法的整体过程, 通过高级语言进行学习以及实现; 通过资料学习其他分组密码, 例如 DES 加密过程的 S 盒的差分分析, 来了解差分分析的具体过程, 并且借助高级语言将数学公式转换为可以自动执行大容量数据加密的差分的实际应用当中。结合 LBlock 算法对 S 盒, 去对每一个 S 盒进行输出差分以及相对应对可能的输入明文对进行汇总, 生成表格; 随后根据该表格对一轮的输入的明文拆分后进行差分计算, 每个 S 盒都换取不同的攻击值进行计算, 最终得出当轮使用的 32 位密钥值。再进行多轮的同样的操作, 即可获得更多位数的密钥。

1.1.2 研究意义

密码学是信息安全的基础, 也是整个网络报文, 数据流等传送过程中的一道“屏障”, 对我们的信息交流起着决定性的作用。而分组密码又是密码学中地位非常高的一种类型, 是实现信息加密的核心体制。在现在, 随着物联网的发展以及科技的迅速更新, 越来越多的事物加入到了“互联网”这个大家庭, 高效, 轻便, 易于存储, 运行能耗低成为了新兴时代的代名词。对应的, 加密系统的安全强度也成为了专业领域人们最关注的方面之一。一般情况下, 加密算法会通过芯片等硬件设备进行运行, 在这个过程中就会在不经意间泄漏一些信息, 从而遭遇一些旁路攻击。

我们所研究的 LBlock 算法就是一种新兴的, 高效轻便的分组密码算法, 软

硬件上都有不错的运行功效,因而可用于 RFID 标签盒其他的一些低消耗设备上,进行物联网的信息维护。所以对于现在的我们来说,虽然他不比 DES 等算法广泛被运用,但是依然具有非常高的研究价值。除此之外,差分分析是我们研究分组密码非常有效的攻击方法, LBlock 算法兼具了 Fesitel 结构的加解密结构以及 SPN 结构的代换置换系统。通过差分分析,对算法的中间状态进行研究和攻击,在短时间内可以获取部分内容的密钥信息,为整个网络的安全性都是有非常重要的意义。

1.2 国内外发展现状 (Development Status at Home and Abroad)

1949 年,著名的学家 Shannon 发表的一篇名为《*Communication Theory of Secret System*》中首次提出了分组密码的设计理念,而真正的公开研究分组密码正是在上世纪 70 年代的 DES 的算法的问世。90 年代末,美国发起的 AES 计划正式迎来了分组密码理论的飞速发展阶段。

分组密码中最重要的两个设计原则就是混淆和扩散,目的就是为了防止攻击者进行信息的统计分析来破坏密码系统。因此大多数分组密码都会精心设计这两个环节,利用非线性的 S 盒实现混乱,利用线性变换达到扩散的目的。在 20 世纪 70 年代之前,对这方面的研究还是微乎其微的,理论研究非常的滞后。在 DES 算法问世之后,民间也刮起来一阵对分组密码研究的狂潮,使得分组密码的理论分析日益的成熟。同时,对他们的差分分析和线性分析也逐渐的多起来。

90 年代 Biham 等人发表了针对 DES 算法的差分分析研究,在此之后,密码学届的人开始使用差分法对当代所有的分组密码进行了安全性分析。1993 年, Matsui 在 *EUROCRYPTO* 上发表了对 DES 算法的线性密码分析的结果。慢慢地, DES 算法的 16 轮加密对差分分析和线性分析都是不免疫的。

很快,在 1997 年美国国家技术标准研究所发起了一场对称密码算法的活动,即 AES 计划。1998 年, NIST 宣布接受了 15 个候选分组密码算法,并请全世界范围内的密码学家来协助分析这些算法的安全性。之后, 15 个算法中又挑出来 5 个算法进入决赛圈,在 2000 年的时候 NIST 决定使用 *Rijndael* 当作高级加密标准 (AES)。

在我国内,吴文玲院士在 2011 年的 ANCS 会议上代表提出来 LBlock 算法。这一算法由于它在软硬件上的效率和它本身的安全性,引发了国内外许多人的关注,同时也有了各种各样的攻击论文发表。包括吴文玲院士自己就对 LBlock 算法进行过中间相遇攻击^[1]; 18 年的时候李玮等人发表了针对 LBlock 算法的唯

密文攻击^[2]，使用了 6 种区分器对算法进行了分析；同时基于算法的积分攻击^[3]也发展到了构建 8 轮区分器，4 轮高阶积分扩散，再到 6 轮高阶积分扩散的高度；LBloc 算法的相关密钥不可能飞来去器分析^[4]也发展到了攻击 22 轮的进度。而在我国的《密码学报》当中也发表了很多不同类型的差分分析，包括不可能差分^[5]，差分故障分析等；甚至明亚运，祝世雄等人还对 LBlock 算法的扩散层^[6]进行了研究分析，讲述了左移对于整个密文的状态，扩散程度的影响。

1.3 研究内容 (Research Contents)

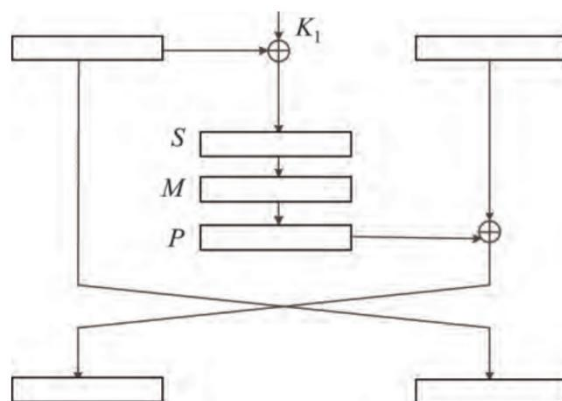


图 1-1 Feistel 单轮结构

Figure 1-1 Feistel single wheel structure

如图所示是常见分组密码的一轮加密结构，其中的 S 盒是用来进行混淆，P 是用来扩散的。我们首先通过研究 LBlock 算法的整体过程，用高级语言实现 SPN 结构^[12]的运算过程，以及密钥扩展函数。在掌握了运行过程的情况下，我们试图去通过添加一些故障来探测算法的扩散程度以及对应位置的变化情况，对每一轮的中间状态进行多次的故障差分，通过 S 盒的代换和 P 函数的扩散，可以找到一条差分路径来获取一定数量的密钥信息，然后再向前继续攻击，恢复的密钥长度就更接近 80 位，以此我们可以在最后通过穷举爆破的方式获取未知位数上的密钥。

1.4 论文组织结构 (Organizational Structure of Papers)

本文的主要研究对象是我国轻量级密码算法 LBlock 分组密码算法以及差分分析法。这个算法完全是自己通过学习其他文献以及 *python* 语言完成编译和运行，后续的差分分析和密钥分析是通过分析自己的运行结果进行阐述的，本文的结构安排如下：

第一章是绪论，简要介绍分组密码的历史发展过程和对于整个加密系统来说

分组密码的地位；还有 LBlock 算法在最近几年的发展与应对各种攻击的效果。最后我们也解释来我们本文的一个研究内容和目标。

第二章介绍 LBlock 分组密码算法的具体流程，详细叙述了 LBlock 每一个流程的细节和部分函数的源代码。

第三章介绍了差分分析的原理，基本概念；还有还给出了 S 盒的一些基本特性，同时介绍了如何制作 S 盒的差分分布表，通过学习范例和 DES 的分布表来制作 LBlock 算法的 10 个差分分布表。

第四章展示整个加密过程和通过差分分布表对多轮的差分攻击结果进行展示和验。最后结合所有的运行结果对密钥进行分析，尽可能多的获取密钥内容。

最后综合全文，总结整个文章的好坏，并对在毕业设计中提供帮助的各位老师表示感谢。

2 LBlock 算法

2 Lblock Algorithm

2.1 算法常见符号 (Common Symbols of Algorithm)

X_k : 明文的第 k 位

C_k : 密文的第 k 位

K_x : 密钥的第 x 位

K^x : 第 x 轮所使用的密钥

$E(x)$: 对 x 进行加密

$D(x)$: 对 x 进行解密

$\Delta X(X_i \oplus X_j)$: 输出的差分

$X \lll i$: X 循环向左移位 i 个比特

$[i]_2$: i 的二进制模式

$C_i \parallel C_j$: C_i 和 C_j 拼接

2.2 算法过程 (Algorithm Process)

在文章中之前我们也提到了该算法加密时，明文长度为 64bit，种子密钥长度为 80bit，应用了 Feistel 网络结构共 32 轮轮函数加密，每一轮函数的密钥都是经过一个函数处理，生成轮 32bit 的子密钥供当轮加密使用，这样一来的话密钥的随机性和不确定性就大大增加。因此，整个加密过程的核心部分就是轮函数中的密钥扩展部分以及对明文两部分对处理。具体加密流程如下图所示：

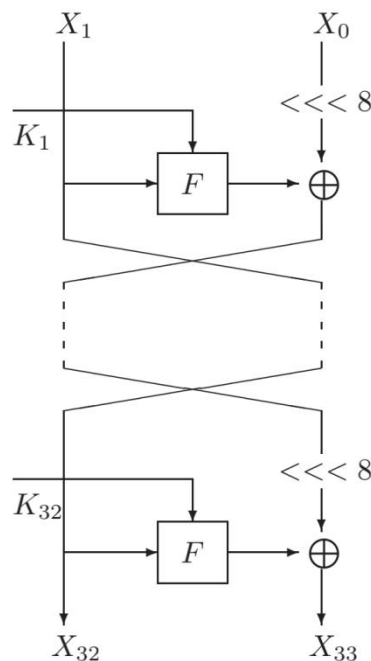


图 2-1 LBlock 加密流程

Figure 2-1 LBlock encryption process

首先将输入的明文切分成各为 32bit 的两半 $X_1 \parallel X_0$ ，每一轮在进行加密处理

过后都会在下一轮加密之前进行左右明文的互换，直到最后一轮加密结束后，最

终密文不需要再进行互换，直接输出 $X_{32} \parallel X_{33}$ 。第 n 轮的轮函数加密结果都可以用公式表示为： $C_n = X_{n-1}$ ， $C_{n+1} = F(f(k_n) \oplus X_{n-1}) \oplus (X_{n-2} \ll 8)$

2.2.1 轮函数中的 S 盒

在密码学中，S 盒^[7](Substitution-box)是对称密钥算法执行置换计算的基本结构。而 S 盒在所有分组密码结构当中都是唯一的非线性部件。S 盒的作用就是将原先的输入打乱并将该位置的数据扩散到其他位置，使得对明文的猜测变得困难。LBlock 算法当中共有 10 个 S 盒，前 8 个 S 盒是针对 32 位明文处理的，后两个 S 盒 S_8 和 S_9 是在密钥扩展处理过程使用的。下表即为 LBlock 算法中的 S 盒内容。

表 2-1 LBlock 算法 S 盒
Table 2-1 Substitution boxes of LBlock Algorithm

S_0	14,9,15,0,13,4,10,11,1,2,8,3,7,6,12,5	S_5	2,13,11,12,15,14,0,9,7,10,6,3,1,8,4,5
S_1	4,11,14,9,15,13,0,10,7,12,5,6,2,8,1,3	S_6	11,9,4,14,0,15,10,13,6,12,5,7,3,8,1,2
S_2	1,14,7,12,15,13,0,6,11,5,9,3,2,4,8,10	S_7	13,10,15,0,14,4,9,11,2,1,8,3,7,5,12,6
S_3	7,6,8,11,0,15,3,14,9,10,12,13,5,2,4,1	S_8	8,7,14,5,15,13,0,6,11,12,9,10,2,4,1,3
S_4	14,5,15,0,7,2,12,13,1,8,4,9,11,10,6,3	S_9	11,5,15,0,7,2,9,13,4,8,1,12,14,10,3,6

我们拿 S_0 盒为例来解释 S 盒的用法。对于 LBlock 算法，S 盒的处理为“4 进 4 出”的模式，即， $\{0,1\}^4 \rightarrow \{0,1\}^4$ 。如果输入的是 1001(9)，那么对应的输出应为 $S_0[9]=2(0010)$ 。

相应的，在一轮加密的过程中，32bit 的明文按顺序地划分为 8 个 4bit 的小块，和对应的 S 盒进行处理。紧接着在 S 盒处理之后，LBlock 又引入了一个扩散层 P 层的概念，它就是将原先顺序的明文打乱顺序，将不同位置的数据扩散到其他位置。表示公式以及流程图如下：

$$P = S_7(X_7) \parallel S_6(X_6) \parallel S_5(X_5) \parallel S_4(X_4) \parallel S_3(X_3) \parallel S_2(X_2) \parallel S_1(X_1) \parallel S_0(X_0)$$

$$C = P_6 \parallel P_4 \parallel P_7 \parallel P_5 \parallel P_2 \parallel P_0 \parallel P_3 \parallel P_1$$

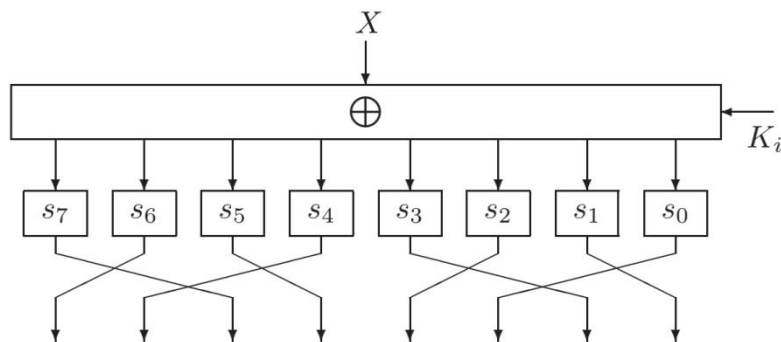


图 2-2 S 盒处理过程

Figure 2-2 processing of S-box

2.2.2 轮函数中的密钥扩展函数

和 S 盒变换当中有一个相近的地方就是所有的数据角标均为由大到小排列。种子密钥 K 表示为 $K = k_{79}k_{78}k_{77}k_{76} \dots k_3k_2k_1k_0$ 。每一轮会选择前 32 位作为当轮的子密钥，之后再通过一些处理将密钥进行打乱，具体流程和代码如下：

1. $K_i = K_i \lll 29$; 首先将密钥整体循环左移 29 位
2. $\{k_{79}k_{78}k_{77}k_{76}\} = S_9(k_{79}k_{78}k_{77}k_{76})$, $\{k_{75}k_{74}k_{73}k_{72}\} = S_8(k_{75}k_{74}k_{73}k_{72})$; 将前 8 位进行 S 盒处理
3. $\{k_{50}k_{49}k_{48}k_{47}k_{46}\} = \{k_{50}k_{49}k_{48}k_{47}k_{46}\} \oplus [i]_2$; 将中间 30-35 位与当轮轮次的二进制进行异或
4. $Ck^i = k_{79}k_{78}k_{77}k_{76} \dots k_{48}$ 。

```
def gen_key(key,num):
    shift_key=""
    ret_key=""
    res=""
    if num==1:
        return key
    if num!=1:
        shift_key=left_turn(key,29)

ret_key=sBoxConfusion(shift_key[0:4],
9)+sBoxConfusion(shift_key[4:8],8)+shi
ft_key[8:]
```

```
print(" 密钥经过 S 盒变换后：",ret_key)

res = str(bin(num)[2:])
for gz in range(0, 5 - len(res)):
    res = '0' + res
tmp=ret_key[30:35]
tmp=str_xor(tmp,res)
ret_key=ret_key[:30]+tmp+ret_key[35:]

#print(" 密钥经过异或后：",ret_key)

return ret_key
```

3 差分分析

3 Differential Analysis

分组密码的差分分析^[8]方法是起源于上个世纪 90 年代,是由 Biham 和 shamir 在一次国际密码年会上针对 DES 算法提出的一种选择明文攻击,从此掀起了一波对于差分分析研究的热潮。同时,在当时的密码学杂志《*Journal of Cryptology*》在 1991 以专刊的方式刊登了 Biham 和 Shamir 的想法,之后的 1993 年 *Spring* 出版了他们两个人的专著,介绍了 DES 密码相关的差分分析应用。

差分分析密码的方法是攻击迭代行分组密码最有效的方法之一,也是衡量它是一种选择明文攻击的方法。其基本思想时通过分析特定明文差分对结果密文差分的影响来获得可能性最大的密钥。差分密码分析的原理和公式其实并不难懂,发展至今,该方法也出现了许多的变种,例如不可能差分,差分与线性分析并用等方法,但是万变不离其宗,都是通过计算来得到一个概率较高的密钥的可能值。接下来我们会介绍差分分析的基本概念以及差分分析针对 LBlock 算法的实际应用。

3.1 差分分析基本概念 (Basic Concepts of Differential Analysis)

差分值(*difference*): 假设 X 和 X' 均为长度为 n 的数据流, 即, $X, X' \in \{0,1\}^n$, $\Delta X = X \oplus X'$, 则称 ΔX 为它们两个的差分值。

差分对(*differential*): 假设一队明文对 $(X, X') \in \{0,1\}^n$, 并且它们的差分值 $X \oplus X'$ 为 m , 再经过 i 轮加密之后, X, X' 对应的输出密文分别是 Y, Y' , 则令 $Y \oplus Y'$ 的值为 n , 那么, 我们称 (m, n) 为该密码分组的一个 i 轮差分对。若 $i=1$ 时, 称 m 为加密时的输入差分, n 为该加密的输出差分。

差分特征: 迭代分组密码的一条 i 轮差分特征 $\delta = (\beta_0, \beta_1, \beta_2, \dots, \beta_{i-1}, \beta_i)$ 是指当两个输入的差分值满足 $X \oplus X' = \beta_0$, 并且各自在中间状态的每一轮加密后的差分值也满足 $Y \oplus Y' = \beta_j$, 且 $1 \leq j \leq i$ 。当 $i=1$ 时, 差分 (α, β) 和差分特征 (β_0, β_1) 概念统一起来表示该轮函数的差分传播特性。

差分特征(*difference characteristic*)也称作是差分路径(*difference path*)。差分仅仅定义了输入和输出的差分值, 在他们中间状态的差分值并未给定, 但是差分特征还指定了中间状态的差分值。通常来说, 寻找一套差分特征要比寻找一条差分更容易, 因为当给定输入差分时, 攻击者必须跟踪轮函数的迭代过程, 才能最终达到某个输出差分, 而完成这一过程的同时也相当于寻找了一条差分特征。

正确对与错误对: 给定一条 i 轮差分特征 $\delta = (\beta_0, \beta_1, \beta_2, \dots, \beta_{i-1}, \beta_i)$, 如果输入对 (X, X') 在中间加密的过程中中间状态的差分值满足差分特征, 则称为该输入对 (X, X') 针对给定的差分特征是一个正确对, 否则称其为错误对。这个概念的引入, 能够更好地理解和估计差分密码分析的数据复杂度。

3.2 S 盒的差分分布表 (Differential Distribution Table of S-box)

3.2.1 S 盒基本介绍

S 盒的首次出现是在 Lucifer 算法当中, 该算法起源于美国 IBM 公司 Watson 实验室在 1970 年研制出的用于数据加密的分组密码体制, 也是我们所熟知的 DES 算法的前身。S 盒是许多分组密码算法当中唯一的非线性部件。因此, 它的复杂程度很大意义上决定轮整个密码算法的安全强度, 同时它的效率也决定了整个加密过程中中间状态密文的混乱程度。S 盒的本质上是看作一种映射, 一种 $S(x) = (f_1(x), f_2(x), f_3(x), \dots, f_n(x))$, 通常我们在描述 S 盒时会以 $n \times m$ 的形式写出, 也就是说当成一种矩阵的形式。当 n 和 m 都是非常大的时候, 即 S 盒元素数量也非常大的时候, 它的线性程度就越低, 攻击时使用的统计特性也就越无迹可寻了。不过同时的, 开发者在设计大规模 S 盒时也就越困难, 因为设计者在研究如何能最大限度的非线性混乱时, 也要兼顾在加解密时的效率, 如同 Lblock 算法当中加解密的顺序是完全可逆的, 其中的 S 盒在加解密过程中的作用也是可逆的。

现在分组密码类型逐渐的增多, S 盒设计的准则主要有: 非线性度、差分均匀度、代数次数及项数分布、雪崩效应、扩散特性以及相关免疫性。SPN 结构的密码还要求 S 盒具有可逆性, 也就是我上文中所提到的 LBlock 算法正是具有这样的特性。

3.2.2 S 盒的非线性度

密码函数的非线性度^[9]是其抵抗线性攻击能力的主要指标, 这个概念最早是在 1988 年由 Pieprzyk 等人提出的。在 $V = \text{GF}(2)$ 是二元 Galois 域, V_n 表示 V 上的 n 维向量空间的前提下, 假设 $f(x)$ 是 V_n 上的一个 n 元布尔函数, 令 $S(x) = (f_1(x), f_2(x), f_3(x), \dots, f_n(x))$ 是 V_n 上的一个多输出函数, 称 N_s 为 $S(x)$ 的非线性度, 它等于 S 盒的各个输出位的非零线性组合所形成的布尔函数与 1 之间的最小汉明距离。汉明距离表示两个 (相同长度) 字对应位不同的数量, 我们以 $d(x, y)$ 表示

两个字 x, y 之间的汉明距离。对两个字符串进行异或运算，并统计结果为 1 的个数，那么这个数就是汉明距离。

在线性密码分析^[10]中，关键的一步在于构造单轮的有效逼近距离，本文主要是关于对密码的差分分析，所以线性分析的内容在这里不做过多阐述。

3.2.3 S 盒差分均匀度以及分析

差分密码分析是目前对于分组密码攻击最有效的方法之一，自从这种方法开始普及之后，开发者就开始对应地研究和构造针对差分分析免疫的 S 盒，并提出一种衡量免疫程度的概念：差分均匀度^[11]。因为差分密码分析主要是利用了 S 盒差分分布矩阵中的特殊元素。

差分分布表的构造就是假设所有元素都属于一个集合 V ，构造一个 $2^m \times 2^n$ 的表格，规则如下：以 α 为行指标遍历 V^m ，以 β 为列指标遍历 V^n ，在他们行列交错位置坐标上的值记作是 $N_s(\alpha, \beta)$ ， α 代表的是输入差分， β 为输出差分，它们三个元素组合在一起就称作是 S 盒的差分分布表。

$N_s(\alpha, \beta)$ 这个值的意义表示为：原本的输入值 $x \in V^m$ ，对于 α 同样 $\in V^m$ ， $\beta \in V^n$ ，那么遍历所有值可以得到 2^m 个输入对，这些输入对中能满条件： $S(x) \oplus S(x \oplus \alpha) = \beta$ 的个数为 $N_s(\alpha, \beta)$ 。因为 x 和 $x \oplus \alpha$ 是互为一组解的，所以这个值的个数一定是偶数。而本文中所做的差分分布表是比上述所说的更为直观，也更为明显的能看到差分信息，称为是 $IN_s(\alpha, \beta)$ 。当攻击者观察到一组 S 盒中的输入盒输出差分 (α, β) 时，可以通过表中元素推出有哪些可能的值可以构造出该输入输出差分，这样一来，这一点对于我们所研究的低轮次的攻击会有非常明显的效果。

S 盒的差分分析模型如下图所示：

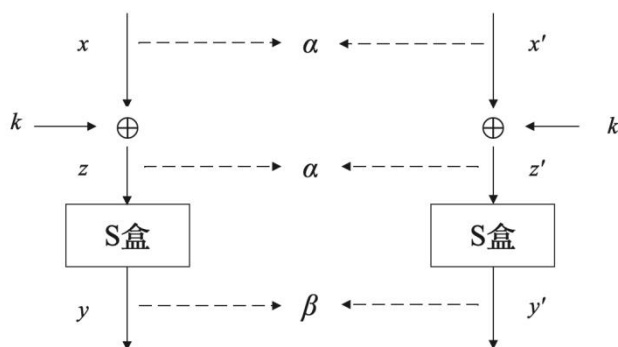


图 3-1 S 盒的差分分析模型

Figure 3-1 Differential Analysis Model of S-box

这里的 x 是明文输入, k 是密钥, 以 4bit 为单位, 每一个 S 盒的输入为 $x \oplus k$ 。我们在攻击某一轮的密钥时, 在知道前一轮加密后的中间状态以及本轮加密后的状态的条件下, 此时根据一组 (x,y) , 即可获得 2 个或者 4 个可能的输入值。以下 10 个表就是自己通过计算总结出的 LBlock 算法中 10 个 S 盒对应的输入输出差分对应的差分分布表。

表 3-1 S_0 盒差分分布表
Table 3-1 Differential distribution table of S_0 -box

输出差分 输入差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	7,6 c,d		8,9				0,1		4,5 e,f		a,b				2,3
2	2,0 b,9		d,f				4,6		1,3 8,a		c,e				5,7
3		8,b c,f				1,2 4,7				9,a d,e				0,3 5,6	
4			0,4	9,d a,e	2,6	8,c b,f					3,7		1,5		
5				1,4 2,7	9,c		8,d			0,5 3,6			a,f		b,e
6		1,4 2,4		0,6 3,5	d,b		9,f						8,e		a,c
7			1,6	b,c 8,f	0,7						2,5		3,4	a,d 9,e	
8		5,d	3,b			6,e	2,a			4,c	1,9			7,f	0,8
9			5,c				7,e	1,8 3,a			4,d	0,9 2,b			6,f
A	4,e f,5	3,9				0,a				1,b			6,c 7,d	2,8	
B	3,8 a,1							5,e 4,f				6,b 7,c	2,9 0,b		
C		a,6	2,e		3,f	5,9		7,b	0,c			4,8			1,d
D			7,a		5,8			0,d	6,b	2,f		3,e		1,c	4,9
E		0,e			4,a	3,d	5,b	2,c	7,9		6,8	1,f			
F					1,e		3,c	6,9	2,d	7,8	0,f	5,a		4,b	

表 3-2 S_1 盒差分分布表
Table 3-2 Differential distribution table of S_1 -box

输入差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	8,b c,f	e,f 4,5	a,b				2,3			6,7 c,d	8,9				0,1
2		1,3 8,a	c,e				5,7			9,b 0,2	d,f				4,6
3					1,2 7,4				9,a d,e				0,3 5,6		
4			3,7	9,d a,e	8,c b,f	1,5					0,4			2,6	
5				1,4 2,7		a,f	b,e		0,5 3,6					9,c	8,d
6	1,7 2,4			0,6 5,3		8,e	a,c							b,d	9,f
7			2,5	8,f b,c		3,4					1,6		a,d 9,e	0,7	
8	6,e		0,8		5,d		1,9		7,f		2,a		4,c		3,b
9			6,f				4,d	2,b 0,9			7,e	1,8 a,3			5,c
A	0,a	6,c 7,d			3,9				2,8				1,b	4,e 5,f	
B		2,9 0,b						b,d 7,c				4,f 5,e		1,a 3,8	
C	5,9		1,d		6,a	0,c		4,8		3,f		7,b			2,e
D			4,9			6,b		3,e	1,c	5,8		0,d	2,f		7,a
E	3,d				0,e	7,9	6,8	1,f		4,a	5,b	2,c			
F						2,d	0,f	5,a	4,b	1,e	3,c	6,9	7,8		

表 3-3 S_2 盒差分分布表
Table 3-3 Differential distribution table of S_2 -box

输出差分 输入差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	a,e 9,d	4,5 b,f				6,7 c,d				a,b	2,3			8,9	0,1
2	1,4 2,7	1,3 8,a				9,b 0,2				c,e	5,7			d,f	4,6
3	0,6 3,5							8,b f,c	1,2 4,7			9,a d,e	0,3 5,6		
4	8,f b,c		1,5				2,6		8,c b,f	3,7				0,4	

5	a,f			$9,c$				$6,e$	$0,5$ $3,6$		$8,d$
6	$8,e$			b,d	$2,4$ $1,7$			a,c			$9,f$
7	$3,4$			$0,7$			$2,5$		a,d $9,e$	$1,6$	
8					$6,e$	$5,d$	$0,8$	$1,9$	$7,f$	$4,c$	$2,a$ $3,b$
9		$0,9$ $2,b$	$1,8$ $3,a$				$6,f$	$4,d$		$7,e$	$5,c$
A	$7,d$ $6,c$			$4,e$ $5,f$	$0,a$	$3,9$			$2,8$	$1,b$	
B	$2,9$ $0,b$	$6,d$ $7,c$	$4,f$ $e,5$	$3,8$ $1,a$							
C		$0,c$	$4,8$	$7,b$	$3,f$	$5,9$	$6,a$	$1,d$			$2,e$
D		$6,b$	$3,e$	$0,d$	$5,8$			$4,9$	$1,c$	$2,f$	$7,a$
E		$7,9$	$1,f$	$a,4$		$3,d$	$0,e$	$8,6$			$5,b$
F		$2,d$	$5,a$	$1,e$				$0,f$	$4,b$	$7,8$	$3,c$

表 3-4 S_3 盒差分分布表Table 3-4 Differential distribution table of S_3 -box

输出差分 输入差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	$0,1$ a,b		$2,3$ $8,9$		e,f		c,d						$6,7$		$4,5$
2	$5,7$ c,e		$4,6$ d,f		$8,a$		$9,b$						$1,3$		$0,2$
3				$8,b$ c,f		d,e $a,9$						$0,3$ $5,6$		$1,2$ $4,7$	
4					$3,7$		$0,4$	$9,d$ a,e	$1,5$		$2,6$	$8,c$ b,f			
5						$1,4$ $2,7$		$0,5$ $3,6$	b,e		$8,d$		a,f		$9,c$
6				$3,5$ $0,6$				$2,4$ $1,7$	a,c		$9,f$		$8,e$		b,d
7					$1,6$		$2,5$	b,c $8,f$	$0,7$		$3,4$			a,d $9,e$	
8				$2,a$	$4,c$	$3,b$	$6,e$					$1,9$	$5,d$	$0,8$	$7,f$

9	b,f $4,d$			$2,b$		$3,a$			$5,c$ $7,e$			$0,9$		$1,8$
A	$2,8$ $3,9$			$4,e$		$6,c$				$0,a$ $1,b$	$7,d$		$5,f$	
B	b,d $4,f$	$3,8$ $2,9$							$1,a$ $0,b$	$7,c$ $5,e$				
C		$0,c$	$7,b$	$1,d$	$5,9$			$4,8$	$3,f$		$2,e$			$6,a$
D		$a,7$	$1,c$		$0,d$	$5,8$		$2,f$	$4,9$				$6,b$	$3,e$
E		$5,b$	$0,e$	$7,9$			$1,f$	$3,d$	$8,6$		$4,a$	$2,c$		
F		$1,e$	$5,a$			$0,f$	$7,8$	$6,9$	$2,d$			$4,b$	$3,c$	

表 3-5 S_4 盒差分分布表Table 3-5 Differential distribution table of S_4 -box

输出差分 输入差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	$6,7$ c,d				$4,5$ e,f				$8,9$		$0,1$		a,b		$2,3$
2	$0,2$ $9,b$				$1,3$ $8,a$				d,f		$4,6$		c,e		$5,7$
3								$8,b$ c,f		$1,2$ $7,4$		$9,a$ d,e		$0,3$ $5,6$	
4		$9,d$ a,e	$2,6$				$1,5$		$0,4$	$8,c$ b,f			$3,7$		
5		$1,4$ $2,7$	$9,c$				a,f				$5,d$	$0,5$ $3,b$	$1,9$		b,e
6		$3,5$ $0,6$	b,d				$8,e$	$2,4$ $1,7$			$9,f$		$4,d$		a,c
7		b,c $8,f$	$0,7$				$3,4$		$1,6$					a,d $9,e$	
8								$5,d$	$3,b$	$6,e$	$2,a$	$4,c$	$1,9$	$7,f$	$0,8$
9				$1,8$ $3,a$		$0,9$ $2,b$			$5,c$		$7,e$		$4,d$		$6,f$
A	$4,e$ $5,f$						$6,c$ $7,d$	$3,9$		$0,a$		$1,b$		$2,8$	
B	$1,a$ $3,8$			$5,e$ $4,f$		$7,c$ $6,b$	$2,9$ $0,b$								
C			$3,f$	$7,b$	$0,c$	$4,8$		$6,a$	$2,e$	$5,9$					$1,d$

D	5,8	0,d	6,b	3,e	7,a	2,f	1,c	4,9
E	4,a	2,c	7,9	1,f	0,e	3,d	5,b	6,8
F	1,e	6,9	2,d	5,a	3,c	7,8	0,f	4,b

表 3-6 S_5 盒差分分布表
Table 3-6 Differential distribution table of S_5 -box

输出差分 输入差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	4,5 e,f				a,b		2,3		6,7 c,d				8,9		0,1
2	1,3 a,8				c,e		5,7		0,2 9,b				d,f		4,6
3				8,b c,f		1,2 4,7						9,a d,e		0,3 6,5	
4		a,e 9,d	1,5		3,7	8,c b,f					2,6		0,4		
5		1,4 2,7	a,f				b,e				9,c	0,5 3,6			8,d
6		3,5 0,6	8,e	2,4 1,7			a,c				b,d				9,f
7		b,c 8,f	3,4		2,5						0,7		1,6	a,d 9,e	
8				6,e	0,8	5,d	1,9					7,f	2,a	4,c	3,b
9					b,f		4,d	0,9 2,b		1,8 3,a			7,e		5,c
A	6,c 7,d			0,a		3,9					4,e 5,f	2,8		1,b	
B	0,b 2,9							7,c 6,d		4,f 5,e	3,8 1,a				
C			0,c	5,9	1,d	6,a		4,8	3,f	7,b					2,e
D			6,b		4,9			3,e	5,8	0,d		1,c		2,f	7,a
E			7,9	3,d		0,e	6,8	1,f	4,a	2,c			5,b		
F			2,d				0,f	5,a	1,e	6,9		4,b	3,c	7,8	

表 3-7 S_6 盒差分分布表

Table 3-7 Differential distribution table of S_6 -box

输出差分 输入差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1		$0,1$ a,b	e,f				$6,7$			$2,3$ $8,9$	c,d				$4,5$
2		c,e $5,7$	$8,a$				$1,3$			$4,6$ d,f	$9,b$				$0,2$
3	$8,b$ c,f				$0,3$ $5,6$				$9,a$ d,e				$1,2$ $4,7$		
4			$3,7$	$9,d$ a,e	b,f $8,c$	$1,5$					$0,4$			$2,6$	
5				$0,5$ $3,6$		b,e	a,f		$1,4$ $2,7$					$8,d$	$9,c$
6	$3,5$ $0,6$			$1,7$ $2,4$		a,c	$8,e$							$9,f$	b,d
7			$1,6$	b,c $8,f$		$0,7$					$2,5$		a,d $e,9$	$3,4$	
8	$2,a$		$4,c$		$1,9$		$5,d$		$3,b$		$6,e$		$0,8$		$7,f$
9			$2,b$				$0,9$	$4,d$ $6,f$			$3,a$	$5,c$ $7,e$			$1,8$
A	$4,e$	$2,8$ $3,9$			$7,d$				$6,c$				$5,f$	$0,a$ $1,b$	
B		$6,d$ $4,f$						$3,8$ $2,9$				$1,a$ $0,b$		$7,c$ $5,e$	
C	$1,d$		$5,9$		$2,e$	$4,8$		$0,c$		$7,b$		$3,f$			$6,a$
D			$0,d$			$2,f$		$7,a$	$5,8$	$1,c$		$4,9$	$6,b$		$3,e$
E	$7,9$				$4,a$	$3,d$	$2,c$	$5,b$		$0,e$	$1,f$	$6,8$			
F						$6,9$	$4,b$	$1,e$	$0,f$	$5,a$	$7,8$	$2,d$	$3,c$		

表 3-8 S_7 盒差分分布表Table 3-8 Differential distribution table of S_7 -box

输出差分 输入差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1		c,d $6,7$	$8,9$				$0,1$			$4,5$ e,f	a,b				$2,3$
2		$0,2$ $9,b$	d,f				$4,6$			$1,3$ $8,a$	c,e				$5,7$

3	8,b c,f			1,2 4,7			9,a d,e			0,3 5,6						
4	2,4 1,7	0,4	a,e 9,d	8,c b,f	2,6				3,7					1,5		
5			1,4 2,7		9,c 8,d		0,5 3,6							a,f b,e		
6			0,6 3,5		b,d 9,f									8,e a,c		
7		1,6	b,c 8,f		0,7				2,5				a,d 9,e	3,4		
8	5,d	3,b		6,e		2,a	4,c		1,9		7,f			0,8		
9		5,c			7,e	1,8 3,a			4,d	0,9 2,b				6,f		
A	3,9	4,e 5,f		0,a			1,b				2,8			6,c 7,d		
B		3,8 1,a				5,e 4,f				7,c b,6				2,9 0,b		
C	6,a	2,e		5,9	3,f	7,b	0,c		4,8					1,d		
D		7,a			5,8	0,d	2,f	6,b		3,e	1,c			4,9		
E	0,e			3,d	4,a	5,b	2,c		7,9	6,8	1,f					
F					1,e	3,c	6,9	7,8	2,d	0,f	5,a	4,b				

表 3-9 S₈ 盒差分分布表Table 3-9 Differential distribution table of S₈-box

输入差分	输出差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1		4,5 e,f	a,b				6,7 c,d	8,9				2,3				0,1
2		1,3 b,8	c,e				0,2 9,b	d,f				5,7				4,6
3	8,b c,f					9,a d,e				1,2 4,7				0,3 5,6		
4			3,7						9,d a,e	8,c b,f	1,5					2,6
5						0,5 3,6		0,4	1,4 2,7		a,f	b,e			9,c	8,d

6	2,4 1,7							0,6 3,5	8,e	a,c			b,d	9,f
7			2,5				1,6	b,c 8,f	3,4			a,d 9,e	0,7	
8	6,e		0,8		7,f		2,a		5,d		1,9		4,c	3,b
9			6,f	0,9 2,b			7,e				4,d	1,8 3,a		5,c
A	0,a	6,c 7,d			2,8				3,9				1,b	4,e 5,f
B		0,b 2,9		7,c 6,b								5,e 4,f		3,8 1,a
C	5,9		1,d	4,8		3,f			6,a	0,c		7,b		2,e
D			4,9	3,e	1,c	5,8				6,b		0,d	2,f	7,a
E	3,d			1,f		4,a	5,b		0,e	7,9	6,8	2,c		
F				5,a	4,b	1,e	3,c			2,d	0,f	6,9	7,8	

表 3-10 S₉ 盒差分分布表Table 3-10 Differential distribution table of S₉-box

输出差分 输入差分	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1				6,7 c,d	4,5 e,f							8,9	a,b	0,1	2,3
2				0,2 9,b	1,3 8,a							d,f	c,e	4,6	5,7
3								8,b c,f	9,a d,e	1,2 4,7	0,3 5,6				
4		9,d a,e				2,6	1,5		0,5 3,6	8,c b,f		0,4	3,7		
5		1,4 2,7				9,c	a,f							8,d	b,e
6		3,5 0,6				b,d	8,e	2,4 1,7						9,f	a,c
7		b,c 8,f				0,7	3,4				a,d 9,e	1,6	2,5		
8								5,d	4,c	6,e	7,f	3,b	1,9	2,a	0,8
9	1,8 3,a		0,9 2,b									5,c	4,d	7,e	6,f

A			4,e 5,f		6,c 7,d	3,9	1,b	0,a	2,8	
B	5,e 4,f	7,c 6,b	3,8 1,a		2,9 0,b					
C	7,b	4,8	0,c	3,f	6,a		5,9		2,e	1,d
D	0,d	3,e	6,b	5,8		2,f		1,c	7,a	4,9
E	2,c	i,f	7,9	4,a	0,e		3,d		6,8	5,b
F	6,9	5,a	2,d	1,e		7,8	4,b	0,f	3,c	

3.2.4 差分分析计算

在进行 S 盒代换时，主要利用了 S 盒的如下关键性质： $(x \oplus k) \oplus (x^* \oplus k) = (x \oplus x^*)$ ，即密钥加不影响输入差分的值。在这种攻击模型下，要恢复部分密钥就需要利用到上节我们写好的差分分布表及其里面的 $IN_s(\alpha, \beta)$ 。值得一提的是，在我们本文中使用的差分故障分析就是以这种方法进行的。我们以 S_0 盒为例子，当输入差分为 e_x 时， S_0 盒的输出差分有 8 种可能的情况，分别是：

$$e_x \rightarrow 1_x, e_x \rightarrow 3_x, e_x \rightarrow 5_x, e_x \rightarrow 6_x, \\ e_x \rightarrow 8_x, e_x \rightarrow a_x, e_x \rightarrow d_x, e_x \rightarrow e_x.$$

相对应的，在表 3-1 中也给出了具体的 $IN_s(\alpha, \beta)$ 取值情况。从正序的角度出发，假设我的密钥值 $k=0111(7_x)$ ，输入的 $x=1001(9_x)$ ， $x^*=0011(3_x)$ ，则加密流程为：

$$x=1001(9_x), k=0111(7_x), y_1=S_0(x \oplus k)=S_0(1110)=1100(c_x).$$

$$x^*=0011(3_x), k=0111(7_x), y_1^*=S_0(x^* \oplus k)=S_0(0100)=1101(d_x).$$

此时，输出差分 $\delta y_1 = y_1 \oplus y_1^* = 1101 \oplus 1100 = 0001(1_x)$ 。

现假设攻击者在攻击过程中获取了这两个输入值 $x=1001(9_x)$ ， $x^*=0011(3_x)$ ，以及最终的一个输出差分为 $\delta y_1=1_x$ ，但并不知道这其中的密钥是多少，因此他希望获取一些密钥 k 的信息。首先我们可以算出输入差分为 $\delta z_1 = z_1 \oplus z_1^* = (x \oplus k) \oplus (x^* \oplus k) = (x \oplus x^*) = 1010(a_x)$ ，根据表 3-1 可获取到集合 $IN_s(\delta z_1, \delta y_1) = IN_s(a_x, 1_x)$ ，此时至少泄漏给攻击者信息如下：

$$a_x \rightarrow 1_x \text{ 当且仅当 } z_1 = x \oplus k \in \{4_x, 5_x, e_x, f_x\}$$

通过这样的计算得到了一组密钥 k 的候选值： $k = x \oplus z_1 \in \{d_x, c_x, 7_x, 8_x\}$ 。

同时，如果攻击者又截获了另一组输入，分别是 $x=1100(c_x)$ ， $x^*=0110(6_x)$ ，但同样只能观察到最终的输出差分为 $\delta y_1=e_x$ ，此时他同样可以获取密钥信息。注意到这一组的输入差分 $\delta z_1 = a_x$ ，输出差分为 e_x ，根据表 3-1 可获取到集合 $IN_s(\delta z_1, \delta y_1)=IN_s(a_x, l_x)$ ，此时至少泄漏给攻击者信息如下：

$$a_x \rightarrow e_x \text{ 当且仅当 } z_2 = x \oplus k \in \{2_x, 8_x\}$$

通过这样的计算得到了一组密钥 k 的候选值： $k = x \oplus z_2 \in \{7_x, e_x\}$ 。那么通过两个集合的交集我们可以得出此次加密的密钥为 7。所以当攻击者的条件弱化为“只能获取明文以及对应的密文的输出差分”时，也可以获取密钥。但是如果只采用一次差分攻击或者只采用一行差分分布表的话，很可能造成范围太大，不能精准查到具体数值的情况。

4 算法的实现以及攻击

4 The Implementation of Algorithm and Attack

4.1 Lblock 加密演示(Lblock encryption demo)

本文中 Lblock 算法运用的 64bit 的明文和 80bit 的密钥都是使用与 MD5 算法类似的“补 0”操作，在文章中使用的明文为字符串“liyifan”，密钥为字符串“lihongan”，首先将两个字符串转换为二进制类型，随后在字符串后面填 0 进行补齐，效果如下图 4-1 所示。

```

请输入信息输入字符串不能为空:
liyifan
please input your key:
lihongan
明文转换为二进制: 01101100011010010111100101101001011001100110000101101110 ( 56 )
密钥转换为二进制: 01101100011010010110100001101110110110011001110110000101101110 ( 64 )
明文填充之后为: 0110110001101001011110010110100101100110011000010110111000000000 ( 64 )
密钥变为: 01101100011010010110100001101110110110011001101100001011011100000000000000 ( 80 )
密钥填充之后为: 0110110001101001011010000110111011011001100111011000010110111000000000000000 ( 80 )

```

图 4-1 加密前对数据的处理

Figure 4-1 Data processing before encryption

现在我们将 32 轮轮函数的子密钥，密文中间状态以表格的形式现实出来，方便于后面章节中对于密文密钥的分析的使用。加密时间为 3.432 毫秒。

表 4-1 轮函数加密的中间状态

Table 4-1 Intermediate State of Round Function Encryption

使用的 32 位密钥为: 01101100011010010110100001101111	
第 1 轮	密钥经过扩展函数处理之后: 0011010011001100111011000010110111100000000000000001101100011010010110100001101 本轮加密后的密文为: 1101111110111100111010100000001001101100011010010111100101101001
使用的 32 位密钥为: 00110100110011001110110000101101	
第 2 轮	密钥经过扩展函数处理之后: 110000100000000000000000110110001111001011010000110100110100110011001110110000101 本轮加密后的密文为: 000101111011000001101100000100101101111101111001110101000000010
使用的 32 位密钥为: 110000100000000000000000110110001	
第 3 轮	密钥经过扩展函数处理之后: 000000101011010000110100110100110101001110110000101110000100000000000000110110 本轮加密后的密文为: 0100001010111001010000101110101000010111101100000110110000010010
使用的 32 位密钥为: 00000010101101000011010011010011	
第 4 轮	密钥经过扩展函数处理之后: 10011001011101100001011100001000100000000000011011000000010101101000011010011010 本轮加密后的密文为: 0000101010001110000100001111111001000010101110010100001011101010
使用的 32 位密钥为: 10011001011101100001011100001000	
第 5 轮	密钥经过扩展函数处理之后: 01011000000000001101100000001010011100001101001101010011001011101100001011100001 本轮加密后的密文为: 01011000010101111000000000110001000010101000111000010000111111110
使用的 32 位密钥为: 01011000000000001101100000001010	
第 6 轮	密钥经过扩展函数处理之后: 0111000100011010011010100110010100011000010111000010101100000000001101100000001 本轮加密后的密文为: 0011001111001110010011011111001101011000010101111000000000110001
使用的 32 位密钥为: 01110001000110100110101001100101	
第 7 轮	使用的 32 位密钥为: 01110001000110100110101001100101

	密钥经过扩展函数处理之后:	00010101000010111000010101100000111000110110000000101110001000110100110101001100
	本轮加密后的密文为:	0001000001101000010110111101010000110011110011100100110111110011
	使用的 32 位密钥为:	00010101000010111000010101100000
第 8 轮	密钥经过扩展函数处理之后:	01010010011011000000010111000101011010011010100110000010101000010111000010101100
	本轮加密后的密文为:	0011111010011101011111100001010100010000011010000101101111010100
	使用的 32 位密钥为:	01010010011011000000010111000101
第 9 轮	密钥经过扩展函数处理之后:	00010100001101010011000001010101000011100001010110001010010011011000000010111000
	本轮加密后的密文为:	0101110111001110111010101111100000111110100111010111111000010101
	使用的 32 位密钥为:	00010100001101010011000001010101
第 10 轮	密钥经过扩展函数处理之后:	00010111110000101011000101001000111100000001011100000010100001101010011000001010
	本轮加密后的密文为:	0101010010011011100000110001101101011101110011101110101011111000
	使用的 32 位密钥为:	00010111110000101011000101001000
第 11 轮	密钥经过扩展函数处理之后:	0101000100000010111000000101000110110100110000010100001011110000101011000101001
	本轮加密后的密文为:	0010011000000100100010001110000001010100100110111000001100011011
	使用的 32 位密钥为:	01010001000000101110000001010001
第 12 轮	密钥经过扩展函数处理之后:	00000000100110000010100001011110100010101100010100101010001000000101110000001010
	本轮加密后的密文为:	0100011100110001101000100110001000100110000001001000100011100000
	使用的 32 位密钥为:	00000000100110000010100001011110
第 13 轮	密钥经过扩展函数处理之后:	101001110101100010100101010000101101010111000000101000000000100110000010100001011
	本轮加密后的密文为:	110111000110111001110111101111101000111001100011010001001100010
	使用的 32 位密钥为:	10100111010110001010010101000101
第 14 轮	密钥经过扩展函数处理之后:	11001101011100000010100000000011101000001010000101110100111010110001010010101000
	本轮加密后的密文为:	010011010001111000011010011001001101110001101110011101111011111
	使用的 32 位密钥为:	11001101011100000010100000000011
第 15 轮	密钥经过扩展函数处理之后:	11011111000101000010111010011100100000101001010100011001101011100000010100000000
	本轮加密后的密文为:	1101100001010111110001000110110001001101000111100001101001100100
	使用的 32 位密钥为:	11011111000101000010111010011100
第 16 轮	密钥经过扩展函数处理之后:	1000100001010010101000110011011111000000101000000001101111000101000010111010011
	本轮加密后的密文为:	11001110110001011111101000001110110100001010111100010001101100
	使用的 32 位密钥为:	10001000010100101010001100110111
第 17 轮	密钥经过扩展函数处理之后:	0110101100010100000000110111110011100001011101001110001000010100101010001100110
	本轮加密后的密文为:	1111101000101110001111100010000111001110110001011111101000001110
	使用的 32 位密钥为:	01101011000101000000000110111110
第 18 轮	密钥经过扩展函数处理之后:	11100001000101110100111000100011000010101000110011001101011000101000000001101111
	本轮加密后的密文为:	010100011110011001001010111001111111010001011100011111000100001
	使用的 32 位密钥为:	11100001000101110100111000100011
第 19 轮	密钥经过扩展函数处理之后:	1001011101010001100110011010111000110000000011011111100001000101110100111000100
	本轮加密后的密文为:	100110110000101111111111000011101010001111001100100101011110011
	使用的 32 位密钥为:	10010111010100011001100110101110
第 20 轮	密钥经过扩展函数处理之后:	11100000000000001101111110000110110111010011100010010010111010100011001100110101
	本轮加密后的密文为:	110100101001010101110001011011110011011000010111111111110000111
第 21 轮	使用的 32 位密钥为:	111000000000000011011111110000110

	密钥经过扩展函数处理之后:	10101010101001110001001001011111110011001100110101111000000000001101111110000
	本轮加密后的密文为:	0100110011110101000001011100101011010010100101010111000101101111
第 22 轮	使用的 32 位密钥为:	10101010101001110001001001011111
	密钥经过扩展函数处理之后:	01100010110011001101011110000010110001101111110000101010101001110001001001011
	本轮加密后的密文为:	0011101010111111000010111011111001001100111101010000010111001010
第 23 轮	使用的 32 位密钥为:	01100010110011001101011110000010
	密钥经过扩展函数处理之后:	00101011110111111100001010101000011111000100100101101100010110011001101011110000
	本轮加密后的密文为:	100101010100110011101101011001010011101010111110000101110111110
第 24 轮	使用的 32 位密钥为:	00101011110111111100001010101000
	密钥经过扩展函数处理之后:	101100111000100100101101100010000011001101011110000001010111011111100001010101
	本轮加密后的密文为:	10101111001100010001100010111000100101010011001110110101100101
第 25 轮	使用的 32 位密钥为:	10110011100010010010110110001000
	密钥经过扩展函数处理之后:	1011000001101011110000001010110001011110000101010110011100010010010110110001
	本轮加密后的密文为:	0111110101001110101110110010110010101111001100010001100010111000
第 26 轮	使用的 32 位密钥为:	10110000011010111100000010101100
	密钥经过扩展函数处理之后:	01001010111000010101011011001101011001001011011000110110000011010111100000010101
	本轮加密后的密文为:	1011001101100011100001100100100001111101010011101011101100101100
第 27 轮	使用的 32 位密钥为:	01001010111000010101011011001101
	密钥经过扩展函数处理之后:	000100101001011011000110110000101100111100000010101010010101110000101011011001
	本轮加密后的密文为:	1000000111011100001110000101101010110011011000111000011001001000
第 28 轮	使用的 32 位密钥为:	00010010100101101100011011000010
	密钥经过扩展函数处理之后:	00101100111000000101010100101000000001010101101100100010010101101100011011000
	本轮加密后的密文为:	1000011110011001110010011010111110000001110111000011100001011010
第 29 轮	使用的 32 位密钥为:	00101100111000000101010100101000
	密钥经过扩展函数处理之后:	101110001010101101100100010010011111101100011011000001011001110000001010100101
	本轮加密后的密文为:	1010010010110001011100010010011010000111100110011100100110101111
第 30 轮	使用的 32 位密钥为:	10111000101010110110010001001001
	密钥经过扩展函数处理之后:	00000011011000110110000010110000010000010101010010110111000101010110110010001001
	本轮加密后的密文为:	1010110101100100011110101110011110100100101100010111000100100110
第 31 轮	使用的 32 位密钥为:	00000011011000110110000010110000
	密钥经过扩展函数处理之后:	10111011001010101001011011100001010011011001000100100000011011000110110000010110
	本轮加密后的密文为:	1010110011110011101111011100100110101101011001000111101011100111
第 32 轮	使用的 32 位密钥为:	10111011001010101001011011100001
	密钥经过扩展函数处理之后:	11111100101100100010010000001111100011011000001011010111011001010101001011011100
	本轮加密后的密文为:	1010110011110011101111011100100110111100110100101101011000100011

4.2 一轮差分分析(Differential analysis of single wheel)

4.2.1 攻击思路

由 LBlock 算法的差分 S 盒可知, 在已知中间状态和输出差分的情况下在一个 S 盒是输入中注入两次故障, 也就是说进行两次差分分析就可以获得准确的密钥, 然后通过密钥扩展函数去逆向计算其他位置的密钥。首先进行故障注入, 形

成不同差分输入的流程图 4-1:

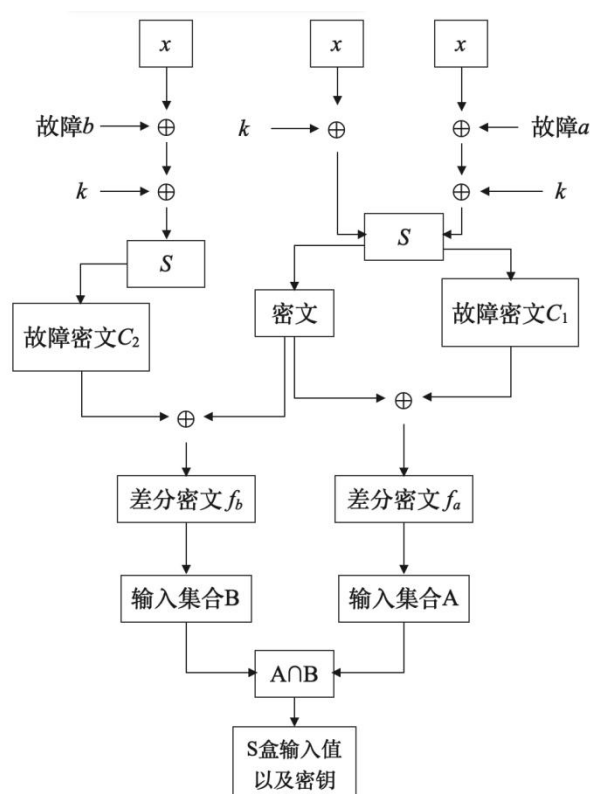


图 4-1 攻击思路流程图

Figure 4-1 Attack thought flow chart

我们首先选取第 32 轮进行攻击，所有的密钥以及密文信息在上表 4-1 中都有体现，攻击流程的文字叙述如下：

- 1) 首先用正确的密钥对正确的明文进行加密，得到正确的密文（但这个过程我们无法得知，只能知道最终正确的密文）。
- 2) 依次对每一个 S 盒所对应的输入位置进行注入故障，得到不一样的输入，这个故障值是随机的，我们使用 *python* 库中的 *random.randint()* 函数来生成。故障长度为 4bit。随后再进行加密得到一个故障密文。
- 3) 将两个密文进行异或得到输出差分 ΔX^{32} 。那么输入差分在文章前段中提到的 $(x \oplus k) \oplus (x \oplus a \oplus k) = a$ ，也就是说，注入的故障也就是输入差分。根据差分表可以得到 2 个或者 4 个备选值。
- 4) 重复上两步可以确定 S 盒的输入。
- 5) 因此可以确定在最后一轮使用的 32 位密钥值。
- 6) 因此可以确定在最后一轮使用的 32 位密钥值。

4.2.2 攻击演示

我们通过对每个 S 盒攻击，一共进攻 8 个回合即可获得一轮密钥，现在我们使用 S_1 盒进行分析。

```

原文文a为: 0x2
密钥经过S盒变换后: 1011111100110111000111001000010110111101000001101111 1101111100011000000001100000
使用的32位密钥为: 10111111001101110001110010000111
real_cipher= 1110011011000100010001000010000001110001011011111000010110000001
输入故障f为: 1111 ( 0xf )
密钥经过S盒变换后: 1011111100110111000111001000010110111101000001101111 1101111100011000000001100000
输入故障f为: 1101 ( 0xd )
密钥经过S盒变换后: 1011111100110111000111001000010110111101000001101111 1101111100011000000001100000

```

图 4-2 S_1 盒差分攻击信息

Figure 4-2 S_1 -box Differential Attack Information

上图所示的就是我们这次攻击的所有信息，框中标注的密钥信息仅为验证攻击成果时使用。我们可以看到已知的原文文输入为 0010，密钥我们未知。两次的故障分别为 0010 和 1101。此时我们可以按照流程来一步一步解析这个过程。因为 Feistel 结构在轮函数加密时会分为左右明文，所以我们在观察输出差分时也在中间进行分开，以免干扰查验。输出差分如下图所示：

```

0000 0000
0000 0000
0000 0000
0000 0000
0000 0000
0000 0000
1111 1101 *
0000 0000
*****
0000 0000
0000 0000
0000 0000
0000 0000
0000 0000
0000 0000
0000 0000
0000 0000
1000 1111 *

```

图 4-3 输出差分

Figure 4-3 Output differential

我们所知道的，轮函数会发生改变的是右边的明文，也就是上图下半部分。可以看到第一个输出差分为 1000，对应的输入差分为故障值本身 1111，经过对比 S_1 盒的差分分布表可知，输入的可能值为 5 和 a 。第二组的输出差分为 1111，输入差分为 1101，对应的可能输入值为 7 和 a ，所以由此可以得到结论，在本轮 S_1 盒里的输入($x \oplus k$)为 a ，由于我们知道明文的输入为 2，密钥 $k=2 \oplus a=1000=8$ ，根据分析，输入 S_1 盒的密钥为 32 为密钥的倒数第二组 4bit 的输入值，验证可得我们攻击有效。下表即为我们针对 32 轮单轮的差分分析的结果（结果均以 16 进制表示），最终再和图 4-2 的框中部分进行验算。

表 4-2 一轮差分分析结果

Table 4-2 Result of a round of Differential Analysis

S 盒	明文值	输入差分 1	输出差分 1	输入差分 2	输出差分 2	交集输入	推测密钥
S ₀ 盒	9	5	7	1	3	8	1
S ₁ 盒	c	1	7	9	8	2	e
S ₂ 盒	d	8	f	c	5	b	6
S ₃ 盒	b	b	2	6	5	2	9
S ₄ 盒	3	2	1	4	2	9	a
S ₅ 盒	f	2	d	3	c	d	2
S ₆ 盒	c	e	1	4	3	7	b
S ₇ 盒	a	1	7	3	5	1	b

我们现在将推测出的密钥进行转化和拼接，来检验是否和表 4-1 中 32 轮所使用的子密钥一致，代码和运行结果如下：



```

3  skey="10111011001010101001011011100001"
4  k=[1,14,6,9,10,2,11,11]
5  res=""
6  for i in range(0,8):
7      x = '{:04b}'.format(k[7-i])
8      res=res+x
9  print (skey)
10 print(res)
11 if res == skey:
12     print("密钥一致")

```

运行结果：

```

/usr/local/bin/python3.9 /Users/ivanlee/Desktop/ivanlee/毕业论文资料/S盒差分代码/yansuan.py
10111011001010101001011011100001
10111011001010101001011011100001
密钥一致

```

图 4-4 密钥验证

Figure 4-4 Key Verification

在 2.2.2 中我们也叙述过了密钥扩展函数的算法，密钥一共长 80 位，我们现在仅仅获得了当轮使用的 32 位密钥，所以我们还要尽可能的还原更多的密钥信息。但是这个子密钥仅仅是在密钥函数运算结束之后截取了前 32 位，所以仅仅是一轮差分分析是对密钥信息的探取用处不大。所以下文我们继续进行第 31 轮的二轮攻击。

4.3 多轮差分分析(Differential Analysis of many Wheels)

4.3.1 第 31 轮攻击演示

表 4-3 二轮差分分析结果

Table 4-3 Result of second round of Differential Analysis

S 盒	明文值	输入差分 1	输出差分 1	输入差分 2	输出差分 2	交集输入	推测密钥
S ₀ 盒	7	e	8	d	3	7	0

S ₁ 盒	<i>e</i>	<i>l</i>	<i>2</i>	<i>e</i>	<i>b</i>	<i>5</i>	<i>b</i>
S ₂ 盒	<i>a</i>	<i>8</i>	<i>e</i>	<i>2</i>	<i>2</i>	<i>a</i>	<i>0</i>
S ₃ 盒	<i>7</i>	<i>9</i>	<i>f</i>	<i>8</i>	<i>c</i>	<i>l</i>	<i>6</i>
S ₄ 盒	<i>4</i>	<i>c</i>	<i>4</i>	<i>d</i>	<i>9</i>	<i>7</i>	<i>3</i>
S ₅ 盒	<i>6</i>	<i>e</i>	<i>6</i>	<i>d</i>	<i>4</i>	<i>0</i>	<i>6</i>
S ₆ 盒	<i>d</i>	<i>3</i>	<i>9</i>	<i>a</i>	<i>l</i>	<i>e</i>	<i>3</i>
S ₇ 盒	<i>a</i>	<i>2</i>	<i>a</i>	<i>a</i>	<i>5</i>	<i>a</i>	<i>0</i>



图 4-5 第二轮密钥验证

Figure 4-5 Key Verification of Second round

4.3.2 第 30 轮攻击演示

表 4-4 三轮差分分析结果

Table 4-4 Result of 3rd round of Differential Analysis

S 盒	明文值	输入差分 1	输出差分 1	输入差分 2	输出差分 2	交集输入	推测密钥
S ₀ 盒	<i>6</i>	<i>6</i>	<i>7</i>	<i>f</i>	<i>b</i>	<i>f</i>	<i>9</i>
S ₁ 盒	<i>2</i>	<i>f</i>	<i>c</i>	<i>2</i>	<i>f</i>	<i>6</i>	<i>4</i>
S ₂ 盒	<i>l</i>	<i>l</i>	<i>2</i>	<i>f</i>	<i>4</i>	<i>5</i>	<i>4</i>
S ₃ 盒	<i>7</i>	<i>l</i>	<i>l</i>	<i>a</i>	<i>b</i>	<i>l</i>	<i>6</i>
S ₄ 盒	<i>l</i>	<i>3</i>	<i>6</i>	<i>5</i>	<i>2</i>	<i>a</i>	<i>b</i>
S ₅ 盒	<i>b</i>	<i>c</i>	<i>4</i>	<i>f</i>	<i>8</i>	<i>l</i>	<i>a</i>
S ₆ 盒	<i>4</i>	<i>f</i>	<i>d</i>	<i>c</i>	<i>8</i>	<i>c</i>	<i>8</i>
S ₇ 盒	<i>a</i>	<i>7</i>	<i>3</i>	<i>9</i>	<i>8</i>	<i>l</i>	<i>b</i>



图 4-6 第三轮密钥验证

Figure 4-6 Key Verification of third round

4.3.3 第 29 轮攻击演示

表 4-5 四轮差分分析结果

Table 4-5 Result of 4th round of Differential Analysis

S 盒	明文值	输入差分 1	输出差分 1	输入差分 2	输出差分 2	交集输入	推测密钥
S ₀ 盒	f	5	4	9	7	7	8
S ₁ 盒	a	a	9	d	a	8	2
S ₂ 盒	9	5	7	9	f	c	5
S ₃ 盒	c	9	d	e	4	9	5
S ₄ 盒	9	2	1	5	3	9	0
S ₅ 盒	9	c	a	e	3	7	e
S ₆ 盒	7	4	5	6	f	b	c
S ₇ 盒	8	2	a	7	d	a	2

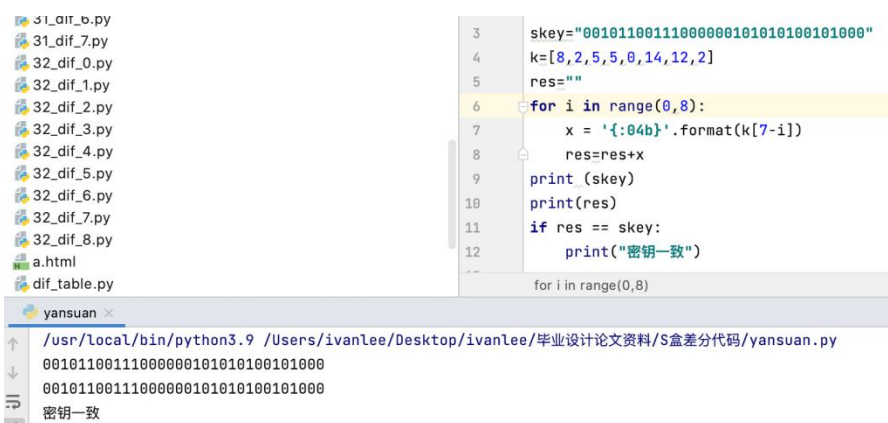


图 4-7 第四轮密钥验证

Figure 4-7 Key Verification of 4th round

4.4 密钥分析 (Key analysis)

现在我们所掌握的密钥信息有如下表所示:

表 4-6 密钥信息

Table 4-6 Key information

第 29 轮:	00101100111000000101010100101000
第 30 轮:	10111000101010110110010001001001
第 31 轮:	00000011011000110110000010110000
第 32 轮:	10111011001010101001011011100001

根据密钥扩展函数，第 31 轮的子密钥 k^{31} 是第 30 轮密钥经过函数处理之后的前 32 位，即 $k^{31}=K^{30}[0:32]$ ，也就是说 $K^{30}=k^{31} || K^{30}[47:0]$ 。

经过函数第一步移位之后密钥 $Key_1=k^{31}[29:31] || K^{30}[47:0] || k^{31}[0:28]$ 。

再之后进行的 S 盒变换之后 $Key_2=S_9(k^{31}[29:31] || K^{30}[47:0]) || S_8(K^{30}[46:43]) || K^{30}[42:0] || k^{31}[0:28]$ 。

加入轮常量 $[i]_2$ 之后， $Key_3=S_9(k^{31}[29:31] || K^{30}[47:0]) || S_8(K^{30}[46:43]) || K^{30}[42:22]) || K^{30}[22:17]) \oplus [31]_2 || K^{30}[16:0] || k^{31}[0:28]=k^{32} || K^{31}[47:0]=K^{31}=k^{32} || K^{31}[47:0]$ 。

我们使用*来代替我们未知的数字，那么 K^{31} 的 80 位内容我们已知的是 80 位其中的后 29 位，为 $k^{30}[0:28]$ ，除此之外其实我们还知道来 k^{32} 的所有内容，为 K^{31} 的 80 位的前 32 位，共计 61 位。

所以我们可以通过倒推 S 盒的方法将 K^{30} 的某几位推出来。 $S_9(k^{31}[29:31] || K^{30}[47:0]) || S_8(K^{30}[46:43])=10111011$ 。由此 $K^{30}[46:43]$ 可以推出为 1000，而 $k^{31}[29:31]$ 我们已经得知是 000，而 $S_9^{-1}(1011)=0001$ ，所以我们 $K^{30}[47]=1$ 。我们继续推测 $K^{30}[42:22]$ 里面的信息，已知 $k^{32}[9:29]=0010101010010110111000$ ，但是 $K^{30}[22:17]) \oplus [31]_2 || K^{30}[16:0]$ 这两部分我们还无法完全得知。 $[31]_2=11111$ ，因为 $k^{32}[30,31]$ 只有两位 01，所以这两位异或结果为 10，还不能推出和 $[31]_2$ 进行异或的后 3 位数具体是多少。

所以我们现在先将 K^{31} 的推测结果如下所示：

101110110010101010010110111000	10***	*****	00000011011000110110000010110
30位, $k^{32}[0:29]$	5位	16位, $k^{30}[16:0]$	29位, $k^{31}[0:28]$

图 4-8 K^{31} 的推测结果Figure 4-8 Conjecture of key K^{31}

我们也可以看到，如果想去解出这个子密钥，已经不需要去话费大量的资源

去试探 80 位，复杂度高达 2^{80} ，相反，我们复杂度经过三轮的差分，从第 30 轮到 32 轮已经探测到了 75% 的内容，剩余部分我们可以继续向前一轮推进。过程依然如下：

$$K^{29} = k^{30} \parallel K^{29}[47:0]$$

$$Key_1 = k^{30}[29:31] \parallel K^{29}[47:0] \parallel k^{30}[0:28]$$

$$Key_2 = S_9(k^{30}[29:31] \parallel K^{29}[47:0]) \parallel S_8(K^{29}[46:43]) \parallel K^{29}[42:0] \parallel k^{30}[0:28]$$

$$Key_3 = S_9(k^{30}[29:31] \parallel K^{29}[47:0]) \parallel S_8(K^{29}[46:43]) \parallel K^{29}[42:22] \parallel K^{29}[22:17] \oplus [30]_2 \parallel K^{29}[16:0] \parallel k^{30}[0:28] = k^{31} \parallel K^{30}[47:0] = K^{30} = k^{31} \parallel K^{30}[47:0]$$

最终我们可以获得 K^{30} 的推测结果：

000000110110001101100000101100	01***	*****	10111000101010110110010001001
30位, $k^{31}[0:29]$	5位	16位, $k^{29}[16:0]$	29位, $k^{30}[0:28]$

图 4-9 K^{30} 的推测结果

Figure 4-9 Conjecture of key K^{30}

所以在 K^{31} 中未知的那 19 位 $(K^{30}[19:17] \oplus 111) \parallel K^{30}[16:0]$ 我们就知道答案为：

$101 \oplus 111 \parallel 0110110010001001 = 010110110010001001$ ，所以第 31 轮的子密钥我们就拿到了，全部的信息如下：

```
000000110110001101100000101100 01010 1101100100010011 0111000101010110110010001001
```

图 4-10 K^{31}

Figure 4-10 K^{31}

4.5 核心代码 (Core Program)

4.5.1 加密代码

```
from S_box import *
from time import *
import re
def str2bin(message):
    res = ""
    for i in message: # 对每个字符进行二
        # 进制转化
        tmp = bin(ord(i))[2:] # 字符转成
        # ascii, 再转成二进制, 并去掉前面的 0b
        for j in range(0, 8-len(tmp)): # 补
            # 齐 8 位
            tmp = '0' + tmp # 把输出的 b
```

给去掉

```
    res += tmp
    #print("转化之后的字符串: ", res)
    return res
# 二进制转化为字符串
def bin2str(bin_str):
    res = ""
    tmp = re.findall(r'.{8}', bin_str) # 每 8
    # 位表示一个字符
    for i in tmp:
        res += chr(int(i, 2)) # base 参数的
    # 意思, 将该字符串视作 2 进制转化为 10 进
```

制

```

    return res
    # 字符串异或操作
def str_xor(my_str1, my_str2): # str, key
    res = ""
    for i in range(0, len(my_str1)):
        # 变成 10 进制是转化成字符串 2
        # 进制与 10 进制异或结果一样, 都是 1,0
        xor_res = int(my_str1[i], 10) ^
int(my_str2[i], 10)
        if xor_res == 1:
            res += '1'
        if xor_res == 0:
            res += '0'
    #print("异或后的字符串: ", res)
    return res
def deal_mess(bin_text):
    ans = len(bin_text)
    if ans % 64 != 0:
        for i in range(64 - (ans % 64)): #
            # 不够 64 位补充 0
            bin_text += '0'
    return bin_text
# 循环左移操作
def left_turn(my_str, num):
    left_res = my_str[num:len(my_str)]
    #left_res = my_str[0:num]+left_res
    left_res = left_res+my_str[0:num]
    return left_res
# 查看密钥是否为 80 位
def input_key_judge(bin_key):
    ans = len(bin_key)
    if len(bin_key) <= 80:
        if ans % 80 != 0:
            for i in range(80 - (ans % 80)):
                # 不够 64 位补充 0
                bin_key += '0'
    if len(bin_key) > 80:
        bin_key = bin_key[:80]
    print("密钥变为: ",
bin_key,"(",len(bin_key),")")
    #print(len(bin_key))
    return bin_key
def gen_key(key,num):

```

```

    shift_key=""
    ret_key=""
    res=""
    """print(key)
    if num==1:
        return key"""
    shift_key=left_turn(key,29)

ret_key=sBoxConfusion(shift_key[0:4],9)+sB
oxConfusion(shift_key[4:8],8)+shift_key[8:]
    #print("密钥经过 S 盒变换后:
",ret_key)
    res = str(bin(num)[2:])
    for gz in range(0, 5 - len(res)):
        res = '0' + res
    tmp=ret_key[30:35]
    tmp=str_xor(tmp,res)
    ret_key=ret_key[:30]+tmp+ret_key[35:]
    #print("密钥经过异或后:
",ret_key)
    print("密钥经过扩展函数处理之后:
",ret_key)
    return ret_key
#S 盒功能: 对 16 比特的字符串进行混淆
def sBoxConfusion(bin_str,num):
    res=""
    tmp = int(bin_str,2)
    ans=S[num][tmp]
    res=str(bin(ans)[2:])
    for gz in range(0, 4 - len(res)):
        res = '0' + res
    return res
def fun_f(bin_Str,key):
    res=""
    z=""
    tmp=str_xor(bin_Str,key)
    for i in range(0,8):
        z+=sBoxConfusion(tmp[i*4:(i+1)*4],7-i)
    #for i in range(0,8):

res=z[4:8]+z[12:16]+z[0:4]+z[8:12]+z[20:24]
+z[28:32]+z[16:20]+z[24:28]
    return res

```

```
def all_message_encrypt(message, key):
    mes_bin=str2bin(message)  #先转换
    成字符串明文
    print("明文转变为二进制:
",mes_bin,"(",len(mes_bin),")")
    key_bin=str2bin(key)      #密钥转换
    成字符串
    print("密钥转变为二进制:  ",
key_bin,"(",len(key_bin),")")
    mes_bin2=deal_mess(mes_bin)  #把
    明文填满 64 位
    print("明文填充之后为:
",mes_bin2,"(",len(mes_bin2),")")
    key_bin2= input_key_judge(key_bin)
    #把密钥填满 80 位
    print("密钥填充之后为:  ",
key_bin2,"(",len(key_bin2),")")
    #subkey=key_bin2
    #key_bin2=gen_key(key_judge)  #
    产生第一轮的密钥
    left_plain=mes_bin2[:32]
    right_plain=mes_bin2[32:]
    shift_plain=""
    for i in range(1,33):
        subkey=key_bin2
        #print(subkey,len(subkey))
        print("第",i,"轮: ")
        mes_tmp=left_plain      #把左半
        边明文存起来
        shift_plain=left_turn(right_plain,8)
    #右半边明文循环左移 8 位
```

4.5.2 差分代码

```
def regular_encode(message,key):
    left_plain=message[:32]
    right_plain=message[32:]
    print("原文 a 为:
",hexhex(left_plain[28:]))
    mes_tmp = left_plain  # 把左半边明文
    存起来
    shift_plain = left_turn(right_plain, 8)  #
    右半边明文循环左移 8 位

    key_f = key[0:32]
    print("使用的 32 位密钥为: ", key_f)
```

```
#print(subkey)
#print(subkey[0:33])
key_f=subkey[0:32]
print("使用的 32 位密钥为: ",key_f)
key_bin2 = gen_key(key_bin2,i)
#产生每一轮的密钥
f_result=fun_f(mes_tmp,key_f)
#轮函数的结果与又明文异或

left_plain=str_xor(f_result,shift_plain)  #
    两边交换
    right_plain=mes_tmp
    if i==32:

left_plain,right_plain=right_plain,left_plain
    print("本轮加密后的密文为:
",left_plain+right_plain)
    return left_plain+right_plain
if __name__ == '__main__':
    print("请输入信息输入字符串不能为
    空: ")
    message = input().replace(' ', '')
    print("please input your key: ")
    key = input().replace(' ', '')
    begin_time=time()
    s = all_message_encrypt(message, key)
    end_time=time()
    run_time=end_time-begin_time
    #out_mess = bin2str(s)
    print("加密过后的内容:" + s)
    print("运行时间为: ",run_time)
```

```
#f_result = fun_f(mes_tmp, key_f)  #
    轮函数的结果与又明文异或
    tmp = str_xor(left_plain, key_f)
    z = ""
    for i in range(0, 8):
        z += sBoxConfusion(tmp[i * 4:(i + 1)
        * 4], 7 - i)
    # for i in range(0,8):
    res = z[4:8] + z[12:16] + z[0:4] + z[8:12]
    + z[20:24] + z[28:32] + z[16:20] + z[24:28]
    righttext = str_xor(res, shift_plain)  #
    两边交换
```

```

    lefttext = mes_tmp
    return lefttext + righttext
def difference_encode(message,key):
    dif=random.randint(1,15)
    guzhang='{:04b}'.format(dif)
    print("输入故障 f 为:
",guzhang,"(",hexhex(guzhang),")")

left_plain=message[:28]+str_xor(guzhang,mes
sage[28:32])#根据 S 盒决定故障注入位置
    right_plain=message[32:]

in_dif=str_xor(right_plain+left_plain,message)
    #print("输入差分为: ",in_dif)
    mes_tmp = left_plain  # 把左半边明文
存起来
    shift_plain = left_turn(right_plain, 8)  #
右半边明文循环左移 8 位
    key_f = key[0:32]
    #print("使用的 32 位密钥为: ", key_f)
    #f_result = fun_f(mes_tmp, key_f)  #
轮函数的结果与又明文异或
    tmp = str_xor(left_plain, key_f)
    z=""
    for i in range(0,8):

z+=sBoxConfusion(tmp[i*4:(i+1)*4],7-i)
    #for i in range(0,8):

res=z[4:8]+z[12:16]+z[0:4]+z[8:12]+z[20:24]
+z[28:32]+z[16:20]+z[24:28]
    righttext = str_xor(res, shift_plain)  #
两边交换
    lefttext = mes_tmp
    return lefttext+righttext

if __name__ == '__main__':
    plain = "My_Plain"
    key = "My_Key"
    real_cipher=regular_encode(plain,key)
    print("real_cipher=",real_cipher)
    dif1_cipher=difference_encode(plain,key)
    dif2_cipher=difference_encode(plain,key)
    xor1_dif=str_xor(dif1_cipher,real_cipher)
    xor2_dif=str_xor(dif2_cipher,real_cipher)

```


5 结论

5 Conclusions

分组密码的设计就是找到一种算法,能在密钥的控制之下从一个足够大盒足够好的置换子集合中简单而又迅速的跳出一个置换,用来对当前的明文进行加密置换。一个好的分组密码系统应该是既难破译,又可以容易实现的。LBlock 算法作为最新一批的分组密码体系,它的 S 盒可以看得出来比 DES 简便了许多,同时他的解密过程也只是加密过程的逆序,说明它的 S 盒在设计的时候就把逆序这一原则考虑的很全面。这一点就非常符合之前我们所说的标准。并且,我们在演示运行结果的过程中,某一位的改变经过多轮的置换,扩散会把这个改变影响到很多的输出。此外,它的安全性盒其他的一些不错的性质必然会让这个算法在今后分组密码体系有更大的作用。

同时,我们此次文章的分析使用的是差分分析,对于攻击类型可以叫做密钥恢复攻击,但其实还有很多其他类型的攻击,在第一章的时候就有所叙述,在我们不知道所有中间状态,或者唯密文攻击的情况下依然可以进行差分分析,这也是我今后值得去学习的地方。

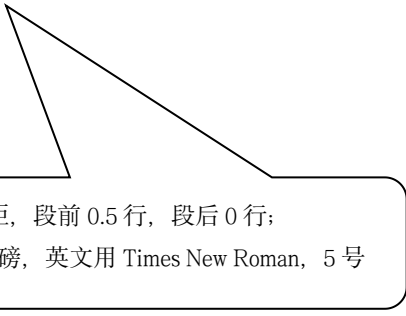
参考文献

- [1] 郑雅菲,吴文玲.LBlock 算法的改进中间相遇攻击[J].计算机学报,2017,40(05):1080-1091.
- [2] 李玮,吴益鑫,谷大武,曹珊,廖林峰,孙莉,刘亚,刘志强.LBlock 轻量级密码算法的唯密文故障分析[J].计算机研究与发展,2018,55(10):2174-2184.
- [3] 潘志舒,郭建胜.LBlock 算法的基于比特积分攻击[J].信息工程大学学报,2013,14(01):30-35.
- [4] 臧越川.轻量级分组密码算法的飞来去器分析研究[D].西安电子科技大学,2015.
- [5] 黄永洪,郭建胜,罗伟.LBlock 算法的相关密钥-不可能差分攻击[J].电子学报,2015,43(10):1948-1953.
- [6] 明亚运,祝世雄,曹云飞.LBlock 结构的扩散层研究[J].计算机工程与应用,2016,52(01):100-104+205.
- [7] 王涛,王永娟,高杨,张诗怡.对轻量级分组密码算法 LBlock 的差分故障攻击[J].密码学报,2019,6(01):18-26.
- [8] 李嘉琪.两种轻量级分组密码的差分故障攻击[D].西安电子科技大学,2020.
- [9] 张雪锋,卫凯莉,姜文.基于 TD-ERCS 序列的 S 盒非线性度优化算法[J].信息安全学报,2021,21(01):10-18.
- [10] 高国强.SM4 抵抗相关密钥线性密码分析的安全性[J].北京印刷学院学报,2020,28(05):154-160.
- [11] 王彦平.利用 Dobbertin 指数构造低差分均匀度函数[J].系统科学与数学,2017,37(04):1156-1165.
- [12] 李艳俊,许星霖.一类 SP 结构的不可能差分区分器证明[J].计算机应用研究,2021,38(05):1529-1532.

翻译部分

英文原文

中文译文



标题：黑体小2加粗居中，单倍行距，段前0.5行，段后0行；
内容：宋体小4号，行距固定值20磅，英文用 Times New Roman，5号

附录 1

煤炭供需预警程序

Imports System.Math

Imports System.Drawing

Public Class Form1

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

With Grid1

.Cols = 9

.Rows = 40

.....

.....

.....