您的理解是正确的。中间相遇攻击（Meet-in-the-Middle Attack）和反弹攻击（Rebound Attack）是两种不同的密码分析方法，它们使用不同的技术和目的：

1. \*\*中间相遇攻击\*\*：

- 这种攻击通常用于已知明文攻击或选择明文攻击的场景，它利用算法的双层结构来减少密钥搜索空间。

- 攻击者通过预计算两个部分的状态，然后在中间状态进行匹配，以此来减少所需的计算量和存储量。

- 中间相遇攻击不依赖于差分分析，而是依赖于算法的结构和预计算的中间结果。

2. \*\*反弹攻击\*\*：

- 反弹攻击是一种差分攻击的变体，它专注于分析算法的差分特性，特别是那些具有复杂S-Box的算法。

- 攻击者通过构造特定的差分特征，并从输出端开始反向追踪，来寻找能够产生这些特征的中间状态。

- 反弹攻击的目的是构造区分器，这种区分器能够在算法的某些轮次中区分加密算法的行为与随机行为，从而减少密钥空间。

总结来说，中间相遇攻击侧重于利用已知明文或选择明文来预计算中间状态，并通过匹配这些状态来破解密钥；而反弹攻击侧重于使用差分分析来构造区分器，从而提高差分攻击的效率。两者都是密码分析中的重要技术，但它们的方法和应用场景有所不同。

Skinny-64

[对分组密码SKINNY.pdf (book118.com)](https://max.book118.com/html/2019/0504/7064125164002023.shtm)

[SKINNY加密算法详解(无代码，仅加密) - 乐悠刘 - 博客园 (cnblogs.com)](https://www.cnblogs.com/leyouliu/p/10665078.html" \l ":~:text=%E5%85%B6%E4%B8%AD%E5%8C%85%E6%8B%AC5%E4%B8%AA%E6%AD%A5%E9%AA%A4%EF%BC%8C%E5%88%86%E5%88%AB%E4%B8%BA %E5%AD%97%E8%8A%82%E6%9B%BF%E6%8D%A2%E3%80%81%E8%BD%AE%E5%8A%A0%E5%B8%B8%E6%95%B0%E3%80%81%E8%BD%AE%E5%8A%A0%E5%AF%86%E9%92%A5%E3%80%81%E8%A1%8C%E7%A7%BB%E4%BD%8D%E3%80%81%E5%88%97%E6%B7%B7%E6%B7%86 %E3%80%82 SKINNY%E6%9C%89%E4%B8%80%E4%B8%8B%E5%87%A0%E7%A7%8D%E6%A8%A1%E5%BC%8F%EF%BC%9A,%E5%85%B6%E4%B8%AD%EF%BC%8Cn%E4%B8%BA%E6%98%8E%E6%96%87%E9%95%BF%E5%BA%A6%EF%BC%8C%E6%9C%8964%E4%BD%8D%E5%92%8C128%E4%BD%8D%E4%B8%A4%E7%A7%8D%EF%BC%9B%E5%AF%B9%E4%BA%8E%E6%AF%8F%E7%A7%8D%E6%98%8E%E6%96%87%EF%BC%8C%E5%AF%86%E9%92%A5%E6%9C%893%E7%A7%8D%E5%BD%A2%E5%BC%8Fn/2n/3n%EF%BC%8C%E5%8D%B364/128%E4%BD%8D%E3%80%8164/128 * 2%E4%BD%8D%E3%80%8164/128 * 3%E4%BD%8D%EF%BC%9B%E5%AF%B9%E5%BA%94%E7%9A%84%E6%A8%A1%E5%BC%8F%E4%B8%8B%EF%BC%8C%E5%88%86%E5%88%AB%E8%BF%90%E8%A1%8C32%E3%80%8136%E3%80%8140...%E8%BD%AE%E3%80%82)

已有中间相遇、Bilicique攻击等，在一篇论文中结合了自动化工具（量子、MILP）改进了反弹攻击，做到15/17轮的分析。(Rebound Attacks on SKINNY hashing with Automatic Tools）（没有很看懂，涉及到多InBound阶段，和经典的反弹攻击不同）

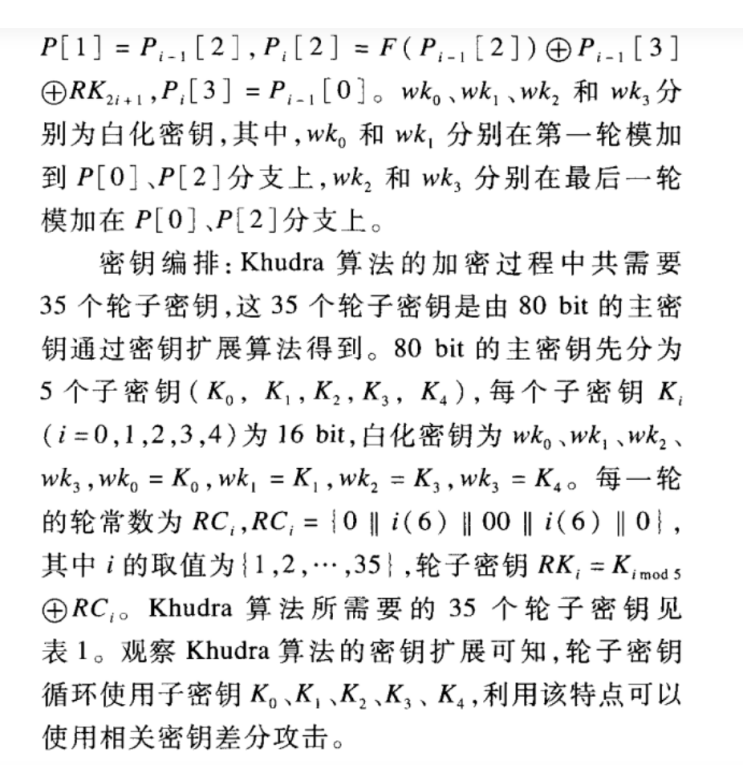
##可以尝试向10轮以外扩展，因为本身只有半字节，看看有没有足够的自由度。

Khudra

应用到Feistel结构：乘积密码指顺序地执行两个或多个基本密码系统，使得最后结果的密码强度高于每个基本密码系统产生的结果.

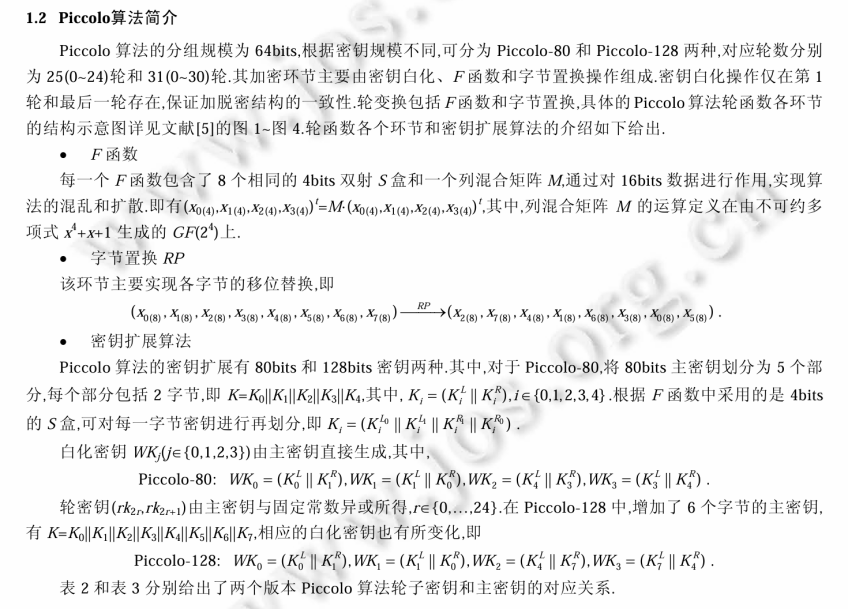
Feistel还提出了实现代换和置换的方法。其思想实际上是Shannon提出的利用乘积密码实现混淆和扩散思想的具体应用。

似乎很难直接使用Rebound Attack



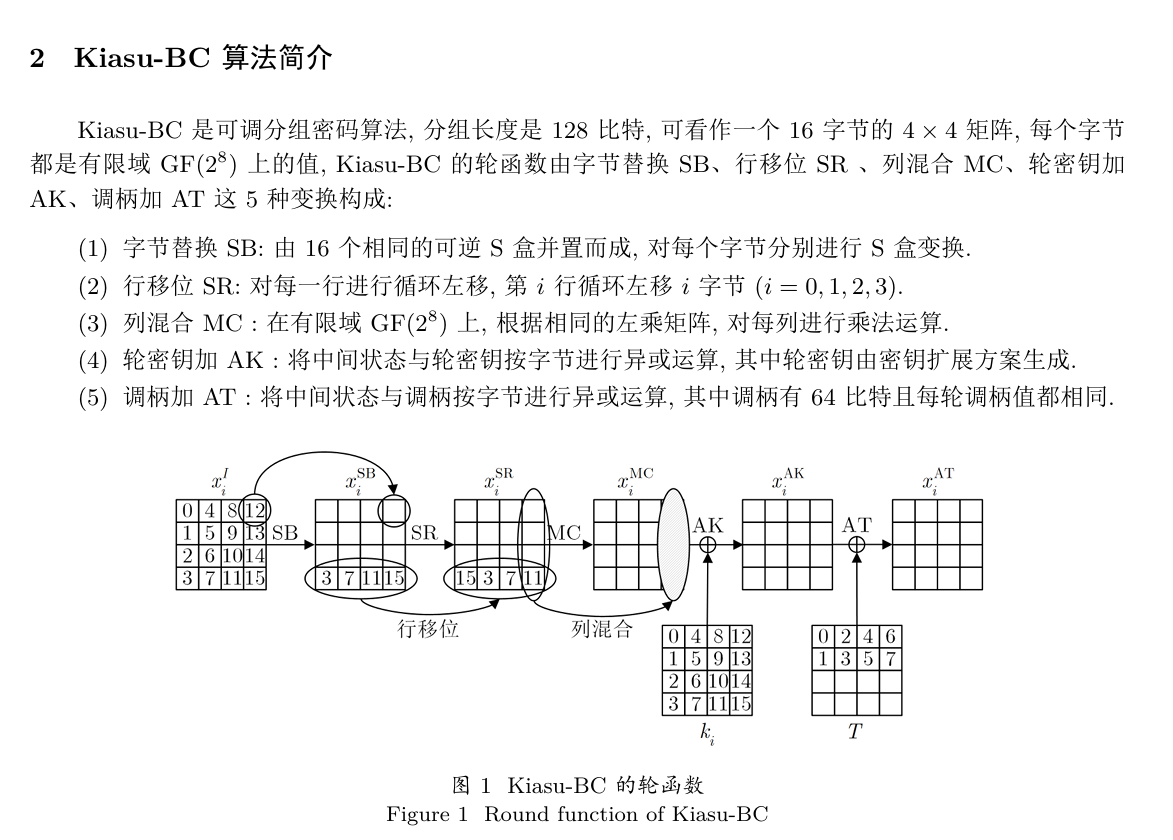
Piccolo

S盒之后的置换比较麻烦，将64比特划分为8个字节进行置换，与经典的路径不相同，可能存在问题。



Kiasu-BC

可以尝试，结构类似于AES，唯一是调柄这个东西需要考量，另外读一读论文中已经有的对这种方法的应用

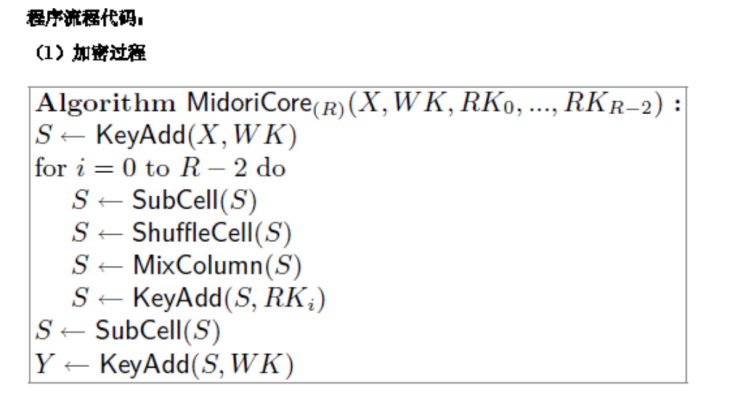


Midori128

[对轻量级分组密码算法Midori的详解\_midori算法-CSDN博客](https://blog.csdn.net/weixin_41596280/article/details/83240077)

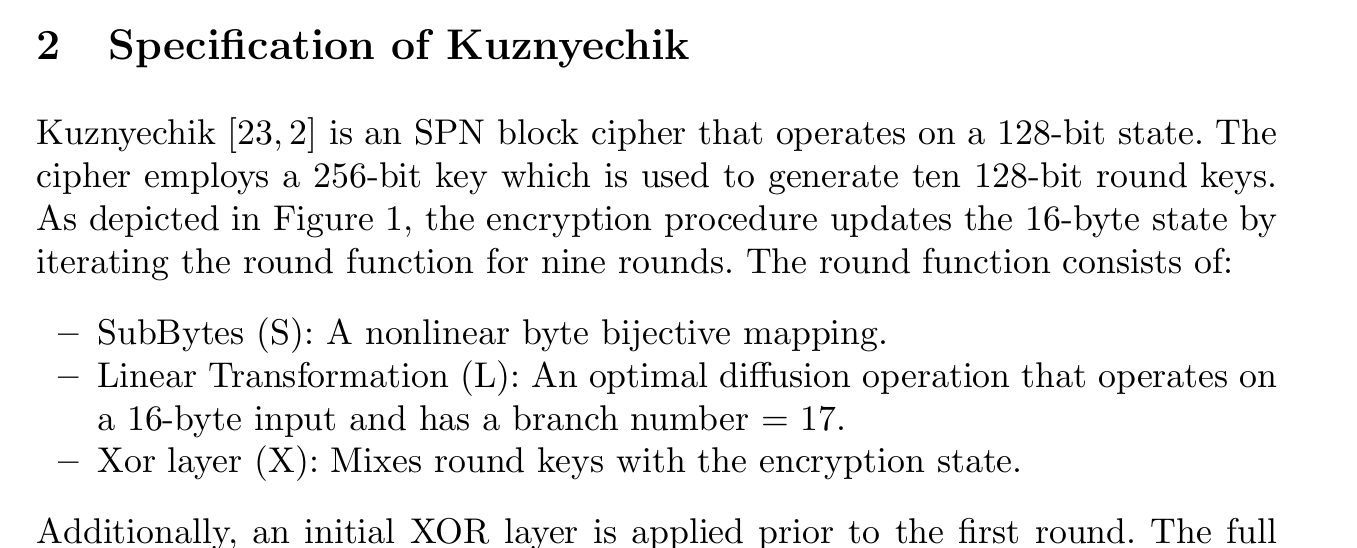
[Impossible Differential Attack on Midori128 Using Rebound-like Technique (iacr.org)](https://eprint.iacr.org/2017/286)

可能性很大，不过有可能已经有一定的研究了，下右图

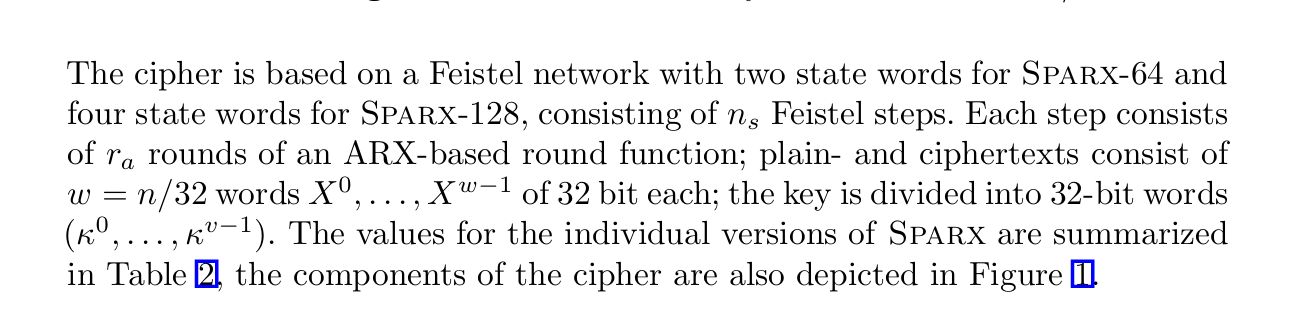


Kuznyechik

已经有中间相遇攻击分析，不过好像没有反弹，看轮函数可能性很大



SPARX-128



用的是Feistel结构，可能不能应用反弹攻击。

Simeck Family

轮函数还不是很清楚，似乎有涉及比特级的置换（旋转），可能不能成功

ChasKey

似乎包含很复杂的置换，可能做不了，是Feistel结构

Crypton

包含比特置换，可能做不了

Joltik-BC和Deoxys

CRAFT

Skinny-64

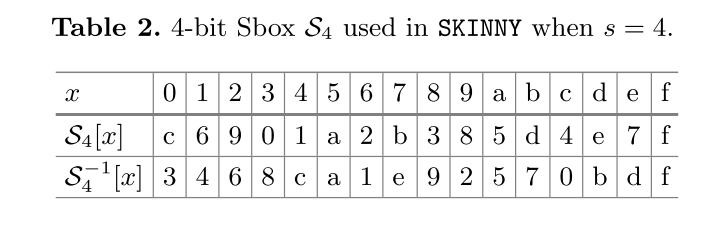
1. MC概率扩散结论

依据古典概率公式：P(4—b) =

所以对半bit的SKinny-64来说，它的MC过程中活跃字节从4到1的概率是2^-10，从4到2的概率是3×2^-7，从4到3的概率是2^-2 。4到固定一点的概率应该是-12次方。

2、4bit的S盒差分匹配性质

SKinny-64的SC非线性替换表如下：



经过试验穷举制作DDT，发现在255个不同的输入/输出差分对中，有159对没有任何解，有72对有一组解，有24组有两组解，加权平均下来解与总差分的比率是120/255≈53.3%，可以认为该S盒也满足差分匹配性质。进而可知同AES类似，中间相遇阶段的平均复杂度也为1。

3、经典反弹攻击

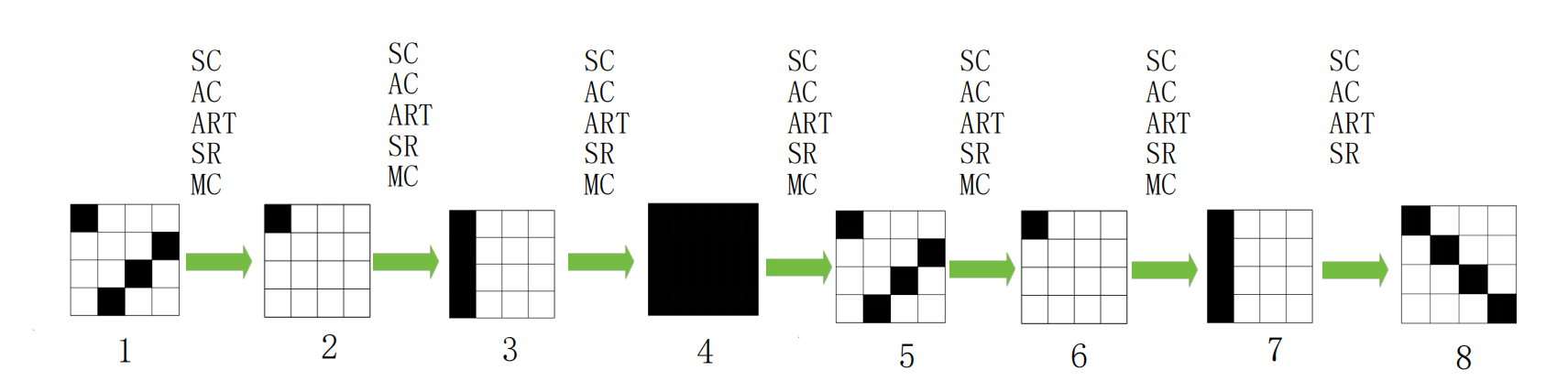
建立如图所示的差分路径，自由度2^16 \* 2^16 = 2^32

就路径本身的可能性而言，这个路径的正向概率是2^(-12-48-12)=2^-72，而路径本身可取到的全体值pairs是(2^4)^16×(2^4)^4=2^80。所以路径本身是可行的。

从首尾各一个活跃字节状态2到6：4轮，概率2^-24，完全可行。如果不考虑到活跃字节的具体位置，就是-20次方概率。

首尾各四个活跃字节区分器状态1到5：4轮，概率2^-24，可行。

首尾各四个活跃字节区分器状态1到8：6.5轮，概率2^-24，可行。



4、超级S盒反弹攻击

轮函数的结构似乎不允许调换SR的顺序，原因在于SR和SC之间夹杂了AC和ART操作，而AES中SR和SB是直接相邻的，因此目前没有找到较好的超级S盒构造方法。

Remark: Skinny-128同样构造上面的路径，SB和MC扩散规律类似于AES结构，最远延伸到6.5轮区分器。

当前已有的研究：

1、Rebound Attack on Skinny hashing with Automatic Tools

这篇论文结合了Mix-Integer Linear Programming (MILP) tool and Constraint Programming (CP) 两种工具，讨论在经典和量子背景下的攻击策略，它采用了Merging Multiple Inbound Phase，其入站阶段可以覆盖3轮，这样的差分路径是自动化搜索得到的，甚至可以匹配到符合路径的秘钥对，实现半自由起始攻击。

成果是实现了128-256，15/17轮的分析，经典条件下时间复杂度2^55.8。

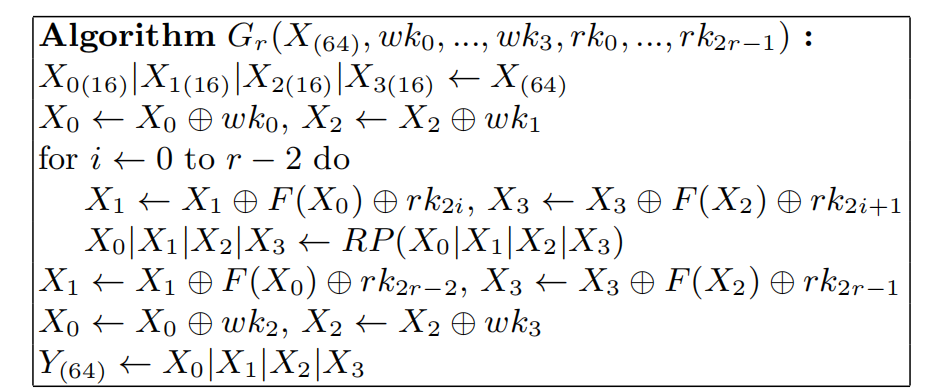
1. 减轮Skinny-128-384算法的中间相遇攻击

构造了Skinny 10.5轮和11轮区分器，并给出了22轮的攻击过程，中间相遇的过程中需要猜测秘钥，秘钥不是已知的。

3、对分组密码 SKINNY- 64的Biclique攻击

这篇论文在Biclique结构的基础上结合中间相遇的思想实现秘钥复现，可以实现全轮匹配，其中时间复杂度约为2^63。

Piccolo



算法和AES距离较远，扩散很难控制，很难直接应用反弹攻击的方式。

Kiasu-BC

1、与AES几乎相同，且调柄对反弹攻击几乎没有影响。

Kiasu-BC算法几乎和AES相同，唯一的不同点在AK轮密钥加后添加了一个Add Tweakey的操作，据资料，Tweakey一般是一个64bit的数据，异或运算在前8个字节上，后8个字节不变，而且初始Tweakey和秘钥扩散算法一般是公开的，所以在寻找碰撞和区分器的过程中可以理解成AK的一部分。已有的，关于AES反弹攻击的结论在Kiasu-BC 上都成立。

当前已有的研究：

1. Chosen-Key Distinguishing Attacks on Full AES-192, AES-256, Kiasu-BC, and More

这篇文章主要用Chosen-Key Related-Key找区分器和碰撞，也用到脸反弹技巧，讨论的轮数从6轮到10轮，时间复杂度在32次方左右。但是路径不完全是反弹攻击，可能还是自动化工具寻找的，另外在Kiasu-BC的讨论中秘钥似乎也有差分，不完全是已知的。这篇对Kiasu-BC、Deoxys-BC等密码可能已经有比较好的结果了，研究价值不大。

2、8轮Kiasu-BC的多重不可能差分攻击

选择明文攻击，缩小候选秘钥范围。

3、Impossible-Differential and Boomerang Cryptanalysis of Round-Reduced Kiasu-BC 和Truncated Boomerang Attacks and Application to AES-based Ciphers

用到了Boomerang方法，选择明文缩小秘钥范围。

（不可能差分这种方法似乎可以用rebound的方式构造contradiction，从而筛选出不可能的秘钥，在多个论文中看到这两种技巧的关联）

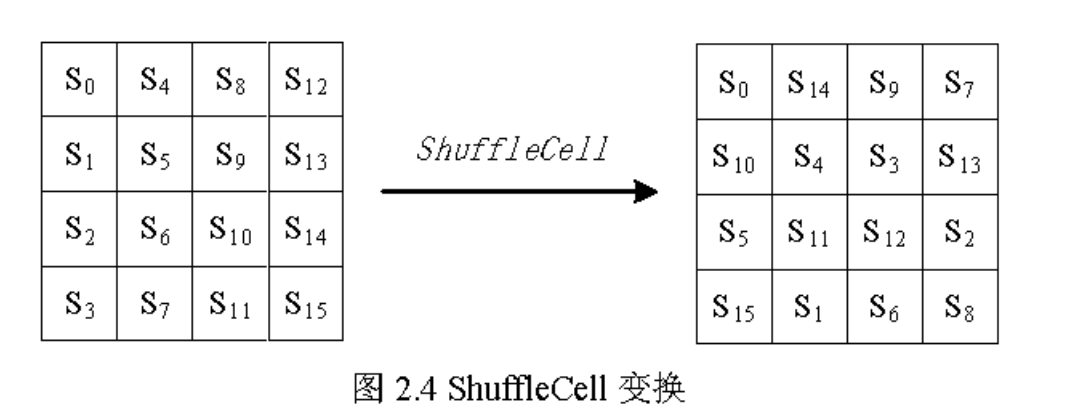
Midori-64

1. Shuffle Cells特性

和AES中的Shift Rows不同，shuffle cells遵循下面的映射：

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

从五个括号中各取一个位置，映射的周期有1、2、4，最小公倍数是4，和Shift Rows本质上一样。也许这种操作有构造新路径的空间，暂时还没有想到好的方法。



1. 反弹攻击

因为SC和Shuffle操作相邻，所以正常的反弹攻击路径和超级S盒都是可行的。类似于Skinny-64中MC性质的讨论，这里的路径的自由度和路径的存在性都与之相同。下面直接引用Skinny-64的部分：

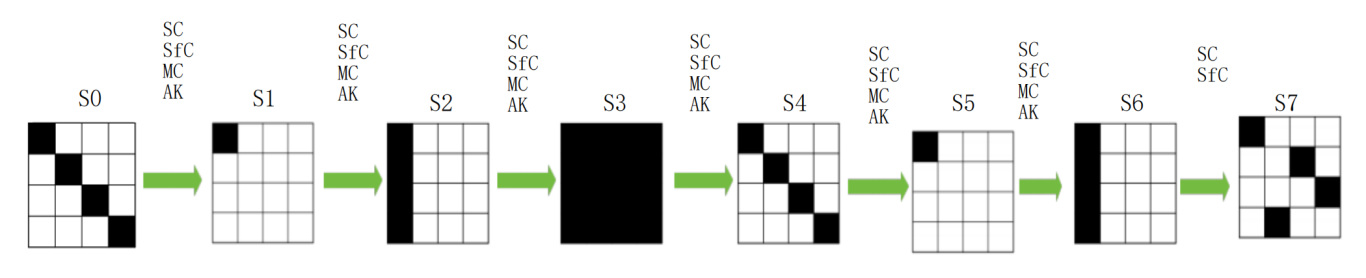
建立如图所示的差分路径，自由度2^16 \* 2^16 = 2^32

就路径本身的可能性而言，这个路径的正向概率是2^(-12-48-12)=2^-72，而路径本身可取到的全体值pairs是(2^4)^16×(2^4)^4=2^80。所以路径本身是可行的。

从首尾各一个活跃字节S1到S5：4轮，概率2^-24，完全可行。如果不考虑到活跃字节的具体位置，则是2^-20。

首尾各四个活跃字节区分器状态S0到S4：4轮，概率2^-24，可行。

首尾各四个活跃字节区分器状态S0到S7：6.5轮，概率2^-24，可行。



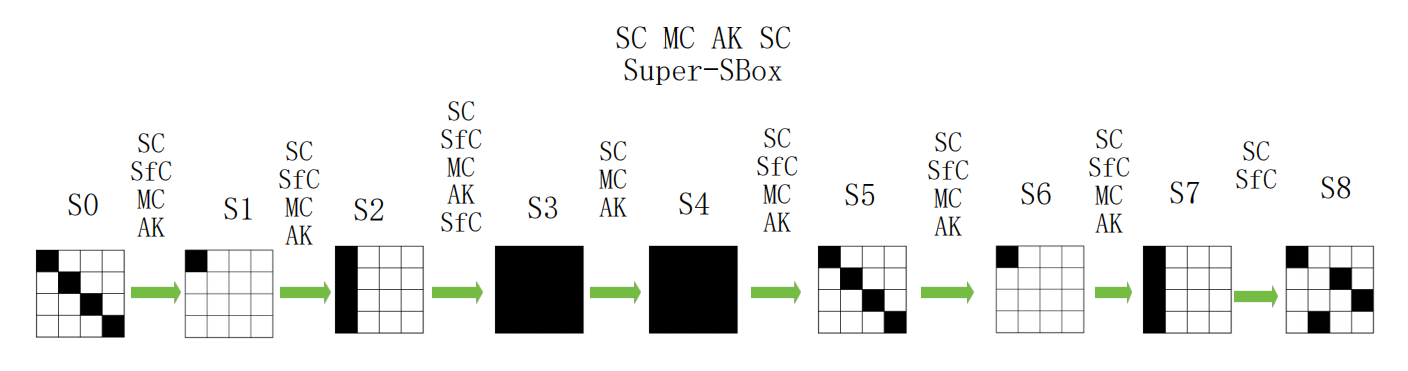
1. 超级S盒

交换第四轮中SC和Shuffle的次序可以构造16bit的超级S盒，扩展一轮，实现7.5轮区分器差分路径。

在制作超级S盒DDT的过程中memory需求是2^16，固定一个输出（或输入）差分，遍历所有可能值得到一个固定输出差分下的，输入差分-解值对映射。

由于OutBound阶段的扩散概率依然是-10次方，中间相遇阶段的平均复杂度为1，所以Inbound阶段运行2^16一般可以获得一个起始点，选择不同的固定差分运行2^20可以获得一个完整匹配路径的解。

（复杂度都比较小，或许可以在笔记本上实现）



当前已有的研究：

1. Midori: A Block Cipher for Low Energy (Extended Version)

这是提出Midori的论文。

1. Meet-in-the-Middle Attacks on Reduced-Round Midori-64

这篇论文应用differential enumeration technique，做的中间相遇，寻找了6轮和10轮区分器。10轮的时间复杂度大约是100次方。

3、Impossible Differential Attack on Midori128 Using Rebound-like Technique

还没有完全理解，主要还是寻找不可能差分路径，然后做秘钥复原。rebound技巧在文中有说明，大概是也需要做匹配表，只不过筛选的是秘钥，然后反弹。

4、New Automatic search method for Truncated-differential characteristics: Application to Midori, SKINNY and CRAFT

论文使用MILP方法寻找差分路径，说这种工具可以应用于很多SPN密码。对标题中提到的三种密码，差分特性存在于6、7、10轮，最大概率是-60，-52，-63次方。

Kuznyechik-128

1. 特殊的Linear Transformation

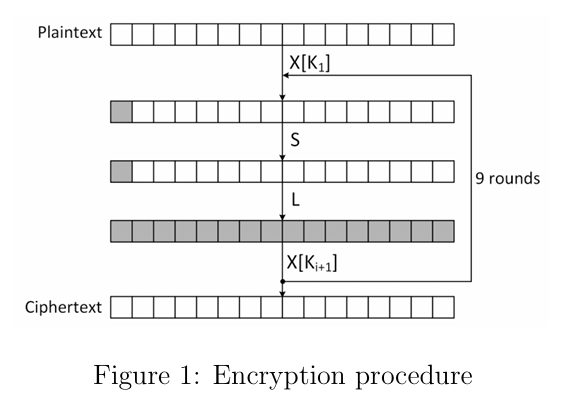
这种密码由字节替换SB，线性扩散LT和秘钥异或相加AK组成，与MC不同，这里的线性扩散是16个字节的列向量左乘一个16阶方阵（在伽罗瓦域上），所以一个活跃字节经过LT之后就会扩散为16个活跃字节。

所以不管初始活跃字节有几位，变换后固定位置1个活跃字节的概率都是2^(-8\*15) = 2^-120，两个活跃字节概率是-112次方，4个活跃字节概率是-96次方，且a+b>=17，a和b是变换前后状态的活跃字节数。

显然如果还按照AES的方式取前后四个活跃字节，自由度无法保证可以求出一组解，所以如果想要在OutBound阶段获得一个字节差分的解，需要至少在InBound阶段引入8个活跃字节，4个字节差分的distinguisher则需要至少前后各6个活跃字节。

对区分器来说，时间复杂度是96次方，正好可以得出一组解，且只能延展到4轮，再往后需要更多的自由度。

因为不是MC，所以应该也无法做出4byte超级S盒。



当前已有研究：

1、A Meet in the Middle Attack on Reduced Round Kuznyechik

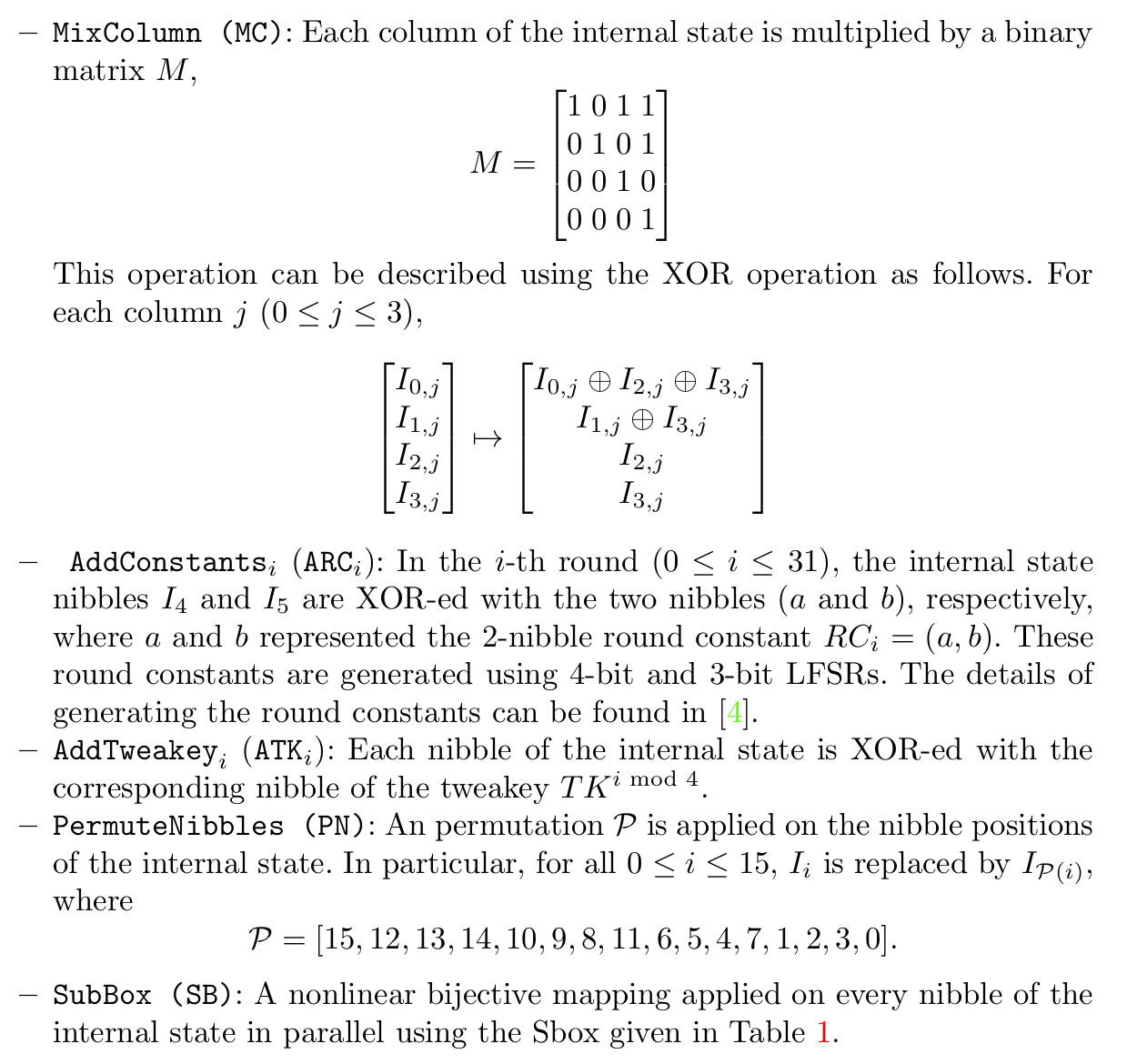
主要结论是5轮秘钥恢复，时间复杂度大概140次方，内存大概150次方。

1. Related-key Attack on 5-Round Kuznyechik

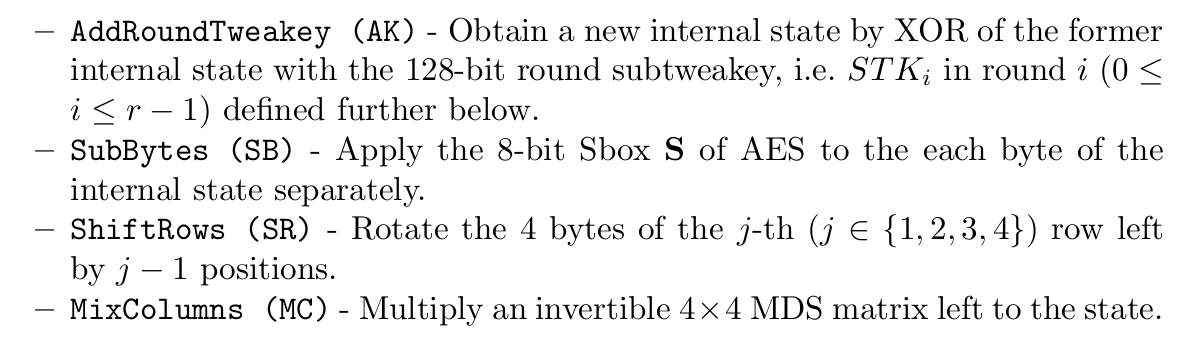
5轮攻击，时间、内存分别是32次方和30次方，related-key有2^16。

其他大同小异的SPN密码

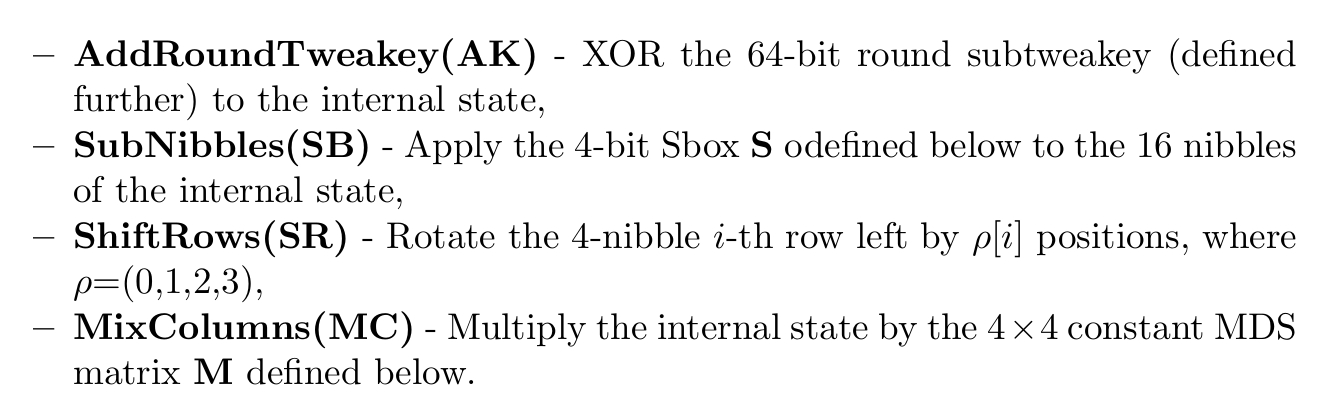
1. CRAFT：轮函数由MC，AC，ATK，P置换(nibble级)和SC组成。



1. Deoxys-BC：轮函数AK，SB，SR，MC



1. Joltik-BC：轮函数ARK，SB，SR，MC



这些函数和AES都非常近似，它们之间主要的不同可能是在调柄或者秘钥的生成方式上，对于known-key的反弹攻击分析，差分路径和复杂度都同上Skinny和Midori的结论。

要继续反弹攻击的研究需要结合其他技术和方法，19年开始有了很多深度学习和差分分析相结合的方法，也许可以实现下面的过程：

在传统的中间相遇阶段中需要匹配16个S盒，但或许可以将这个过程延长成一轮加密+SB操作，传统方法的匹配需要DDT前运算，那么可以用深度学习提取特征的方式来替代DDT的查询匹配作用。比如可以给予大量明文对训练集，输入输出都是差分。训练完成后对输入差分可以先代入模型预测最大概率的一组输出差分，在前向运算得到的差分集合中寻找匹配的，这样可以避免全部经历一轮运算需要的大量计算量。

看看在理论和实际上有没有可能训练出合适的模型。

这是目前大概的思路。

另一个可能的拓展方向是区块链密码安全的保障，后面仔细读一下这两篇论文：[Mathematics | Free Full-Text | Blockchain Interoperability in Data Exchange Logistics Integration (mdpi.com)](https://www.mdpi.com/2227-7390/12/10/1516)

[Research on Logistics Information Blockchain Data Query Algorithm Based on Searchable Encryption | IEEE Journals & Magazine | IEEE Xplore](https://ieeexplore.ieee.org/document/9335619)

结合物联网和区块链，将这个课题引到Data上。

训练集：输入输出差分（实际上还是以明文对的方式给出的）——输入/输出值，目的是构造一个从两端差分（16个cell的图）到16个cell的value的模糊映射（概率）

附录：

1、4bit S盒DDT制作代码

S\_BOX = [0xc, 0x6, 0x9, 0x0, 0x1, 0xa, 0x2, 0xb, 0x3, 0x8, 0x5, 0xd, 0x4, 0xe, 0x7, 0xf]

Inv\_S\_BOX = [0x3, 0x4, 0x6, 0x8, 0xc, 0xa, 0x1, 0xe, 0x9, 0x2, 0x5, 0x7, 0x0, 0xb, 0xd, 0xf]

DDT = [[ [] for \_ in range(16)] for \_ in range(16)]

for i in range(16):

for j in range(16):

out1 = S\_BOX[i]

out2 = S\_BOX[j]

In\_Chafen = i ^ j

Out\_Chafen = out1 ^ out2

DDT[In\_Chafen][Out\_Chafen].append([i,j])

N = 0

N1 = 0

N2 = 0

N3 = 0

for x in DDT:

for y in x:

if y == []:

N = N+1

elif len(y) == 2:

N1+=1

elif len(y) == 4:

N2+=1

elif len(y) == 6:

N3+=1

print(N,N1,N2,N3)