密码学第四次实验报告

# 常规DES算法

## 原理

常规DES算法是基于Feistel密码结构的一种分组密码算法。它通过代替和置换，来代替理想分组密码。在相当长的一段时间里，它的安全性是相当高的。

常规DES算法由于基于Feistel密码结构，需要补全的是每轮的F函数，和16位的密钥。它在Feistel密码结构上也做了扩展，在开始和结束时分别加入了IP和IP-1变换。

子密钥生成算法生成16位的密钥。64位的初始密钥首先去掉奇偶校验位，并通过PC1置换生成56位的实际使用的密钥。然后实际使用的密钥分为左右两部分，每轮中把这两部分分别向左循环移位，再拼接起来，通过PC2置换，得到轮密钥，轮密钥为48位。轮密钥总共生成16个，供16轮使用。

F函数接受32位的输入，通过扩展置换E来把它扩展到48位。然后跟每轮的轮密钥相异或，进入S盒置换，再经过P置换，得到每轮的输出。

Feistel密码结构实际上也是综合了代替和置换。每轮中输入右半部分原样成为输出的左半部分，输入的左半部分与经过F函数处理过的输入的右半部分异或，成为输出的右半部分。由此可见，Feistel密码结构在宏观上确实综合了代替和置换，能够提供逼近分组密码的安全性。

由于DES应用了Feistel密码结构，并在可逆性上做了必要的处理，所以加密算法与解密算法相同，只是解密时使用轮密钥的顺序应该与加密时相反。

## 伪代码

### 单轮Feistel密码结构

**def** feistel\_round(s: BitString, k: BitString,

f: Callable[[BitString, BitString], BitString]) -> BitString:

**return**

### 多轮Feistel密码结构

**def** feistel\_repetitive(s: BitString, key\_list: BitString,

f: Callable[[BitString, BitString], BitString],

rounds: int) -> BitString:

result = s

result = feistel\_round(result, key\_list[i], f)

**return** result

### 密钥生成算法

**def** \_des\_gen\_key\_list(k: bytes) -> List[BitString]:

:

**return**

### 函数

**def** \_des\_f(bs: BitString, k: BitString) -> BitString:

**return** result

### 核心处理算法

**def** \_des(bs: BitString, key\_list: List[BitString]) -> BitString:

result = feistel.feistel\_repetitive(result, key\_list,

\_des\_f, \_des\_round\_count)

**return** result

### 加密算法

**def** encrypt(p: bytes, k: bytes) -> bytes:

key\_list = \_des\_gen\_key\_list(k)

**return** result

### 解密算法

**def** encrypt(p: bytes, k: bytes) -> bytes:

key\_list = \_des\_gen\_key\_list(k)

key\_list.reverse()

**return** result

### 文件加密算法

**def** encrypt\_file(p\_path: str, c\_path: str) -> None:

**while** **True**:

**if** len(b) == 0:

**break**

**if** len(b) < 8:

### 文件解密算法

**def** decrypt\_file(p\_path: str, c\_path: str) -> None:

**while** **True**:

**if** len(b) == 0:

**break**

**if** len(b) < 8:

### 文件加解密演示算法

**def** file\_op\_demo() -> None:

## 分析

### 单轮Feistel密码结构

单轮Feistel密码结构中，F函数的时空复杂度不确定。易知其它操作的时空复杂度为，所以时空复杂度取决于F函数。

### 多轮Feistel密码结构

设轮数为，F函数的时空复杂度分别为和。

单轮复杂度已经求出来，易知总时空复杂度分别为和。

### 密钥生成算法

实际上密钥生成算法并不需要超过级别的额外空间，时间也由于算法规定，复杂度也是。

### 函数

F函数实际上需要一定的额外空间来存放E置换后的结果。但是总体上由于步骤确定、数据规模也确定，时空复杂度也都是。

### 核心处理算法

核心处理算法实际上是多轮Feistel密码结构，以及一些外围的算法。它们的时空复杂度都是，所以总时空复杂度也是。

### 加解密算法

加解密算法实际上是自动生成密钥的核心处理算法。因此易知时空复杂度都是。

### 文件加解密算法

设文件的长度为。

易知对文件的每个块，加解密都是的时空复杂度。由于要填充的块最多只有一个，所以时间复杂度也是。但是由于加密前后数据没有依赖关系，所以空间复杂度是一次加解密的空间复杂度，为。

### 文件加解密演示算法

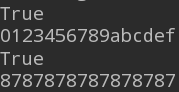
实际上是包装好的文件加解密算法，因此时空复杂度为。

## 测试

使用预置好的几组明密文对进行测试。



经过验算，结果正确。



## 优化

### 单轮Feistel密码结构

实际上单轮Feistel密码结构优化的点比较少，但是可以把要处理的数据的左右两边分别放在一个32位整数中。这样对比特流的异或速度会更快。

### 多轮Feistel密码结构

多轮Feistel密码结构其实数据前后依赖关系非常明显，所以比较难并行化。但是可以让使用的空间变小，这样更能有效地利用处理器的缓存。

### 密钥生成算法

密钥生成算法其实数据依赖比较小，可以并行实现。但是并行实现需要注意并行实体之间的通信、缓存的高效利用和基于时间的侧信道攻击。如果通信没有做好，可能会重新计算密钥。如果使用了基于时间的算法，可能受到基于时间的侧信道攻击。

### 函数

F函数其实在S盒处可以并行化，但是并行化效果不明显，因为处理的数据量太小了。其余的步骤实际上数据依赖比较大，也难以再次分解。

### 核心处理算法

实际上，核心处理算法是F函数和多轮Feistel密码结构的综合。可以把每步的状态信息单独拿出来，再做加密。这样不会保存中间的密钥，但实际上提高不了什么安全性，因为密钥生成的步骤已知，知道任何一步的密钥就能知道所有的密钥。所以为了提高程序的模块化程度，可以提前算出每一步的密钥。

### 加解密算法

加解密算法其实是核心处理算法和密钥生成算法的综合。加解密算法其实只是密钥使用的顺序不一样，因此可以统一，只要在解密时把密钥序列的顺序颠倒过来就行，因此可以把核心处理算法单独拿出来。

### 文件加解密算法

文件加解密其实是使用了把分组密码改造成流密码的一种常见模式：CFB模式。这种模式中每一块的加密是独立的，不依赖于其它的块，因此方便并行执行。但是缺点是明文块和密文块的对应是确定的，所以攻击者能收集到较多的明密文对。

也可以用其它把分组密码改造成流密码的常见模式，它们各有各的特点。

### 文件加解密演示算法

可以在用户友好性上做优化，比如给用户图形界面等等。

# 对三轮DES的差分攻击

## 原理

### 三轮DES算法

三轮DES实际上比常规DES简单得多。常规DES去掉了IP置换和IP-1置换，以及最后的交换左右两半部分，然后把16轮Feistel变换改为3轮，就是三轮DES。实际上三轮DES也可以加上上述的两个置换和交换，但是它们都是很轻易就可逆的，对差分攻击实际上没有什么影响，因此可以忽略。密钥生成算法不变，但是加密时只取前3个密钥。

### 差分

设为一个特定的S盒（），考虑长度为6的比特串的一个有序对，定义的输入异或为，输出异或为。定义集合。因为对任意给定的，确定，也随之确定，因此。对中的每一对，我们能计算出的输出异或，可以计算出分布是非均匀的，这就是攻击DES的基础。

定义，。

### 破译密钥部分比特串

DES中，在第轮中S盒的输入为，其中为的扩展，是由第轮的密钥比特组成。由于，所以输入异或不依赖于密钥比特。

把，和都写成8个6比特串的并：

类似地可表述，和。

对某个，假设我们知道和的值和的输出异或值，那么必有，，。

我们的目的是要破译密钥的部分比特串。定义测试集合。这里是的形式，是固定的，。

假设和为S盒的两个输入，的输出异或为，记，那么密钥比特出现在集合之中。

根据更多的明密文对，能得到更多求出来的，各的交集就是对应所取的可能值。

### 差分攻击

设明文对为和，密文对为和。

有表达式：

类似地，

于是有：

假设选择了明文，使，此时为全0的串且，所以。由于可以从两个明文计算出，可知和已知。有：，知和，其中和分别表示8个S盒的两个输出，而是固定的，为公开已知的置换，因此，由此可知为3轮中8个S盒的输出异或。

另外，和是已知的，它们是密文的一部分。因此，可用公开的扩展函数计算和。对第三轮来说，它们是S盒的输入。于是可由，和，像前面的例子那样对各中密钥比特可能值，着手构造对应的。由他们来确定第三轮密钥中的48位。56位密钥中剩下的8比特可通过穷举种可能来确定。

## 伪代码

**def** des\_3round\_cryptanalysis\_differential(p: List[bytes],

c: List[bytes]) -> List[bytes]:

对每对明文及其相应的密文:

:

计算各的交集

对补全剩下的8位后能得到的所有:

:

**return**

## 分析

设明密文对的个数为。

显然计算，和时空复杂度均为。计算需要遍历个可能的密钥片段，数量一定，因此时空复杂度也均为。类似地，遍历密钥也是的时空复杂度。但是，如果找到了可行的密钥，要检查对明密文。但是数据前后的依赖关系并不大，因此算法总的时间复杂度为，空间复杂度为。

## 测试

采用提前算好的测试数据进行测试。明密文对如下：

|  |  |
| --- | --- |
| 明文 | 密文 |
| 748502CD38451097 | 03C70306D8A09F10 |
| 3874756438451097 | 78560A0960E6D4CB |
| 486911026ACDFF31 | 45FA285BE5ADC730 |
| 375BD31F6ACDFF31 | 134F7915AC253457 |
| 357418DA013FEC86 | D8A31B2F28BBC5CF |
| 12549847013FEC86 | 0F317AC2B23CB944 |

经过计算，得到密钥。



经过验算，结果正确。

## 优化

### 三轮DES算法

实际上三轮DES算法优化空间比较有限，这是因为它不仅数据依赖性比较强，而且把开头和结尾的置换都去掉了。事实上，三轮DES算法代码比较小，能用到现代计算机的缓存机制。

### 差分攻击

差分攻击在寻找密钥的时候能够并行化，而且能够比较好地利用缓存空间。

课件中的算法大致是利用哈希表和计数器，其实也有一种遍历每两对明密文对，然后把不在相应中的密钥片段删除的优化方法。我感觉这种方法如果做成逻辑删除，由于密钥片段只有6位，可以用位运算优化；但是如果真的删除的话，可能由于缓存机制的缘故，效率反而还慢。当然，找可能的密钥和验证密钥是很好并行的。

# 三重DES算法

## 原理

由于单重DES容易被穷举攻击攻破，所以单重DES不再安全。但是，DES提供了很好的抗密码攻击的性能，所以人们不想在短期放弃DES。于是，三重DES被发明出来了。三重DES做三次加密计算，但是只用到两个密钥和。加密算法为，解密算法为。两个密钥确实能提升安全性，但是提升后的安全性不会相当于112位密钥的加密算法。

## 伪代码

### 加密算法

**def** encrypt(p: bytes, k1: bytes, k2: bytes) -> bytes:  
 **return**

### 解密算法

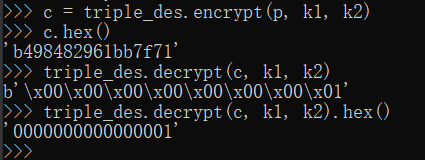
**def** decrypt(c: bytes, k1: bytes, k2: bytes) -> bytes:  
 **return**

## 分析

由于每次执行DES算法的时空复杂度都是，整个算法的时空复杂度也是。

## 测试

令，和分别为0000000000000001，1234132ACDBDEBAA和DEADBEEFDEADBEEF，得到密文和解密结果如下：



可以看出，加解密可逆。经过验算，结果正确。

## 优化

三重DES算法其实我能想到的优化的空间并不大，但是有一点可以优化，就是对生成的轮密钥可以重用。这样并不会减弱算法对侧信道攻击的抵抗能力，因为程序在做什么一般是已知的。

# 对两重S-DES的中间相遇攻击

## 原理

### S-DES

S-DES是简化的DES，明密文长度都为8位，密钥长度为10位，只用两次Feistel结构。虽然它是简化的DES算法，设计目标中有方便人手算这一目标，但是它可以用来研究DES，而且DES有的变换步骤它都有。

### 两重S-DES

两重S-DES就是把明文用两重密钥加密，用到的两个密钥一般不同，但是安全性达不到20位密钥的加密算法的程度，因为有中间相遇攻击。

### 中间相遇攻击

中间相遇攻击是以空间换时间的一种破解多重加密算法的方式，破解两重加密算法更加方便。这是一种已知明文攻击。首先攻击者应该至少知道两对明密文对。然后攻击者的目标是找出和。首先遍历密钥空间，记住所有加密后的结果。然后遍历密钥空间，穷举用所有可能的解密后的结果。如果用某个解密后的结果在所有加密后的结果的集合里，那么攻击者就获得了一堆密钥。如果这对密钥在上也成立，那么就认为找到了密钥对。利用另一明密文对，能够大大减小中间相遇攻击的出错概率。

## 伪代码

### S-DES密钥生成函数

**def** \_sdes\_gen\_key(k: BitString) -> List[BitString]:

:

**return**

### S-DES 函数

**def** \_sdes\_f(bs: BitString, k: BitString) -> BitString:

**return** result

### S-DES核心处理算法

**def** \_sdes(bs: BitStirng, key\_list: List[BitString]) -> BitString:

**return**

### S-DES加密算法

**def** encrypt(p: BitString, k: BitString) -> BitString:

**return** result

### S-DES解密算法

**def** decrypt(c: BitString, k: BitString) -> BitString:

**return** result

### S-DES中间相遇攻击算法

**def** double\_sdes\_cryptanalysis\_mitm(p\_list: List[BitString],

c\_list: List[BitString]) -> Set[BitString]:

:

:

:

:

:

**return** S

## 分析

### S-DES相关算法

跟DES对应算法相似，时空复杂度都是。

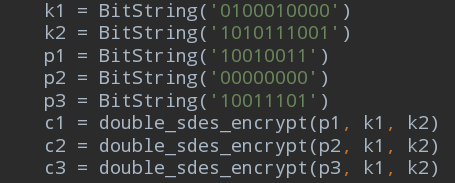
### S-DES中间相遇攻击算法

设明密文对个数为。

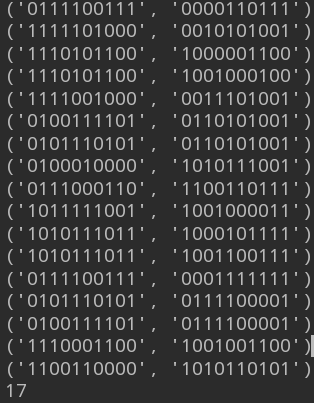
实际上，对第一对明密文对，要加解密的最大次数是有上界的，这个上界不能超过可能密钥个数的二次方，因为所有可能的密钥对个数不能超过它，这样易知时空复杂度都是。但是之后对剩下的对明密文对，可能的密钥对个数会逐渐减少，但是对存放密钥对和其它中间结果的空间的大小需求是不会增加的，因此易知时间复杂度为，空间复杂度为。

## 测试

采用以下的列表、列表、和进行测试。



测试得到所有可能的密钥对。



经过验算，结果正确。

## 优化

### S-DES相关算法

和DES的对应算法一样，在我浅显的认知里，优化算法比较有限。但是DES可以用的优化技巧也可以用过来。

### S-DES中间相遇攻击算法

实际实现存储的机制可以优化。存储可以按照加密后的密文存储，这样第二阶段通过密文找的时候，每次找的时间复杂度就是了。如果第一阶段有多个生成了相同的中间密文，就把它们放在同一个中间密文对应的值下面，用一个列表把所有的记下来。这样第二阶段穷举的时候，时间复杂度会大大降低。其实空间复杂度也降低了，因为用了一种更紧凑的方式，没有把所有的明密文对都记下来。

到了后面测试的时候，由于数据依赖关系较少，并行化性能较好，但是要注意最后筛除不可能的密钥对得到的各个集合应该取交集。

# 总结

## 常规DES算法

我一直听说过常规DES算法，但是这其实是我自己写的第一个正经的加密算法。其实算法并不难，只是繁杂一些，写完了还挺有成就感的。

## 对三轮DES的差分攻击

其实三轮DES是简单的三轮，并没有外面套着的置换。所以这样看，对三轮DES的差分攻击主要是找准数学原理和测试集。实际上找样例评测，之后调试是最难的，真正实现算法真没有那么难，关键是样例有些难找。

## 三重DES算法

这个完全可以并入第一个要求中。只要系统结构实现得好，套三层函数就实现了。

## 对两重S-DES的中间相遇攻击

这其实是另一种攻击方式。选择S-DES做中间相遇攻击的目标，是因为它密钥空间比较小。中间相遇攻击，其实算法都是公开的，实现起来也应该不是很难，主要是验证找到的密钥对需要一些工夫。

## 算法分析与优化

这次虽然算法相对公开，但是优化其实需要花不少精力。中间相遇攻击和对三轮DES的差分攻击，都显示了选择好数据结构的必要性。对三轮DES的差分攻击，其实统计相应密钥片段的出现次数，用哈希表更好，而不是在对应的集合中删除元素，因为哈希表更能体现内存访问的局部性，而且内存地址相对固定，有利于分支预测。中间相遇攻击用第一阶段加密的中间密文作为键更好，因为它是被查找最频繁的字段。这次虽然算法的时空复杂度是常数，但是优化好算法，仍然能优化好算法应用的整体效率。

## 系统设计与维护

这次这几种加密算法的实现，是系统设计的极好例子。这次我在实现各加密算法时，把必要的功能和常量都封装起来，而且把Feistel轮结构单独封装，这样可以更好地复用代码，提高程序的鲁棒性，事实上也加快了我写程序和调试的速度。

## 对课程的建议

感觉我这次花时间比较多的是查找资料，尤其是三轮DES以及S-DES相关，以及样例。感觉老师如果下次能提供一些对应的资料和样例会更好，这样能节省我们的时间。

十分感谢老师能把任务分层，其实我真心地希望老师能推出一个比较明确的评测机制。这是因为高层次的任务需要更多的时间和精力来完成，一般应该有更多的回报。再就是其实应该把更抽象、需要更复杂的算法、更具有技巧性或创造性的任务放到高层，因为这才是智力挑战。像三重DES这种摹仿式的作业，其实应该放在相对较低的层次的。这样可以让分层机制更有效。

## 总结

这次感觉已经到了正经的密码学实验阶段，所以会越来越有挑战性、越来越系统。感觉我的系统化实践终于稍微派上了一点点用场。之后我会继续努力，面对更大的挑战。