密码学第六次实验报告

# 优化的AES算法

## 原理

上一次的实验中，已经可以预想到AES的查表优化。因此这次的优化，可以借助AES的运算性质，通过查表优化的方法来进行。这样可以使AES加密使用到的运算次数大大减小。

具体的查表优化的方法是这样的。对任意一列，有：

其中为字节经过ByteSub部件后的结果，列下标需要模4。

根据每一个完整轮的加密方式，易知上式成立。

因此可以定义4个1024字节的表：

这些表中的取值范围为一个字节的所有可能取值。易知各表的大小确实是1024字i，而且能被事先计算。可以把这些表看成一种函数，这种函数输入一个字节值，产生一个32位的列向量，该向量是对应输入字节的函数。

这样轮函数对一列的操作就能简化成查表和异或：

显然这种方案能大大减小加密操作需要的时间。但是需要的空间略有增加，因此这种方案是一种以空间换时间的方案。

事实上，由于解密算法可以在结构上与加密算法相同，因此类似地可以给出一种对解密轮函数的查表优化方案。同样也是把轮函数对一列的操作简化成查表和异或：

其中为字节经过InvByteSub部件后的结果，列下标也需要模4,是对应的解密变换要查的表。但由于结构上的限制，其中应为经过InvMixColumn部件处理后的轮密钥的第列。

这样，就可以实现加解密的查表优化。

## 伪代码

用到的与AES相关的函数与上次的相应部件实现相同，因此伪代码也相同。但是加密算法和解密算法不同，因此只展示加密算法和解密算法的伪代码，其余函数的伪代码在第五次实验的实验报告中。

### 加密算法

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

由p生成state

由k生成key\_state

key\_list = get\_key\_list(key\_state)

addroundkey(state, key\_list[0])

for i in range(9):

for j in range(4):

得到

把结果与第i轮的轮密钥的第j列相异或，得到下一轮的状态的第j列

用生成的下一轮的状态代替现在的状态

bytesub(state)

shiftrow(state)

addroundkey(state, key\_list[10])

**return** state转换成字节后的结果

### 解密算法

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

由p生成state

由k生成key\_state

key\_list = get\_key\_list(key\_state)

addroundkey(state, key\_list[10])

for i in range(8, 0, -1):

for j in range(4):

得到

把第i轮的轮密钥用InvMixColumn部件处理

把结果与处理后的轮密钥的第j列相异或，得到下一轮的状态的第j列

用生成的下一轮的状态代替现在的状态

invbytesub()

invshiftrow()

addroundkey(state, key\_list[0])

**return** state转换成字节后的结果

## 分析

这种方案实际上用到了一定的空间，因为要存储8个表，每个表的大小为字节，即1KB。但是耗费8KB的存储空间，实际上的开销并不大，因为现代计算机很容易满足8KB的存储需求。这样，每轮只需要4次查表和5次异或运算，虽然时空复杂度仍然为常数级别，但是实际运行效率会大大提高。

## 测试

采用明文54776F204F6E65204E696E652054776F，密钥5468617473206D79204B756E67204675进行测试。得到密文为29C3505F571420F6402299B31A02D73A。这里明文、密文、密钥均采用16进制表示。采用相同的密钥解密密文，得到原来的明文。



可以看出，结果与未经优化的AES算法相同。经过验算，结果正确。

下面比较未经优化与优化后的AES算法的效率，数据仍然选用上文中的数据，重复运行20次得到运行时间和占用内存的平均值。得到结果如下：

|  |  |  |
| --- | --- | --- |
| 算法 | 平均运行时间/s | 占用内存/MB |
| 未经优化的AES算法 | 0.008185 | 0.009019 |
| 优化后的AES算法 | 0.001164 | 0.006789 |

可以看出，经过优化，AES算法的运行效率有了成倍的提升。而且由于减少了中间结果的存放，占用内存也降低了。这就使得AES算法更加实用。

# 总结

## 优化的AES算法

其实这个算法属于非常简单的对加密算法的优化，实际上技术难度并不是很大。重要的一点是弄明白背后的数学原理，这对该算法的实现才是关键的。实际上，数学原理大体就是矩阵运算，因为AES的非线性性并不多。弄明白数学原理以后，实现就相对简单了。

## 算法评估与优化

这次的算法评估与优化，还是沿用了以前的套路。但是，优化的内核越来越偏重数学了。这也是数学对算法优化的一个很典型的例子。

## 系统设计与维护

这次良好的系统设计与维护起到了很好的作用。由于程序进行了模块化处理，所以我可以直接更改相关函数的代码，而且基本上不需要有其他考虑，因为代码结构相对清晰，而且几乎没有荣誉代码。由于上次实验打下的好基础，我可以方便地使用已经有的工具进行调试和逻辑验证，从而加快了实现的速度。同时，由于我以前写过了评测用的代码框架，因此这次评测使用的代码写起来速度能稍微快一些。这应该能体现良好系统设计的加速型回报。

## 对课程的建议

感觉以后可以多强调代码的模块化，这样更能使我们注重代码的可维护性，对以后的实践也是有帮助的。

## 总结

这次实验算一次代码优化实验。这次确实让我能够重视代码优化的重要性，我会在以后的实验中更加关注算法的性能，并且在实现中体现出来。