密码学基础研讨课 实验报告

秦嗣量, 2018K8009970004

January 29, 2021

1 实验内容

用三种不同的方式实现AES-128-ECB加密算法，并测量各种实现的效率。

在此基础上，利用Qt框架提供较为清晰的图形界面，便于直观的检验算法的正确性、更改测试等配置操作，且使得该作品能够直接用于文件的加解密操作中。可方便的加载密钥和明文密文等，并实现加解密操作。

利用标准测试向量验证各种实现的正确性，且在试验中各种实现的结果能够相互验证。

2 实现方法

2.1 基类

将AES加密算法中共性的部分提取出来作为基类，提供统一的接口，底层实现面向该接口编程。此处设计参考了AES-NI指令集的设计。对于每个**128 bits block**的加密，对象内部维护一个**state**，通过更新该成员属性实现加密过程。

密钥扩展 初始化对象时利用提供的参数进行密钥扩展，待后续加解密过程中使用。

加密 利用**xorsi**函数执行初始的密钥加操作，然后利用**encsi**函数执行9轮，最后一轮使用**encsilast**。

解密 利用**xorsi**函数执行初始的密钥加操作，然后利用**decsi**函数执行9轮，最后一轮使用**decsilast**。

设置状态 利用传入的字符串初始化对象内部状态。

获取结果 将内部状态作为字符串导出。

在这样的框架下，各种不同实现的区别仅在于不同内部状态的表示和每个单步操作的实现方式。

2.2 一般实现

逐一实现**sub_bytes**, **shift_rows**, **mix_columns**, **add_round_key**等操作及其逆变换，并拼接以实现上述定义的接口。

2.3 查表法实现

sub_bytes操作后还存在**mix_columns**，需要计算有限域上的矩阵乘法，通过预先计算S盒中的元素乘法运算的结果，可以将各轮变换过程转化为简单的查表、移位和异或操作的组合。可预先计算如下结果:

- Te0[x] = S [x].[02, 01, 01, 03];
- Te1[x] = S [x].[03, 02, 01, 01];
- Te2[x] = S [x].[01, 03, 02, 01];
- Te3[x] = S [x].[01, 01, 03, 02];
- Te4[x] = S [x].[01, 01, 01, 01];

- Td0[x] = Si[x].[0e, 09, 0d, 0b];
- Td1[x] = Si[x].[0b, 0e, 09, 0d];
- Td2[x] = Si[x].[0d, 0b, 0e, 09];
- Td3[x] = Si[x].[09, 0d, 0b, 0e];
- Td4[x] = Si[x].[01, 01, 01, 01];

利用上述几张表加速最终的加密过程。

2.4 AES-NI实现

利用AES-NI指令实现，该指令集中存在与基类中接口一一对应的函数，直接调用即可。

3 环境配置

1. 操作系统：Manjaro Linux （Kernel Version 5.4）
2. IDE: Qt Creator
3. 编译器：gcc 9.3

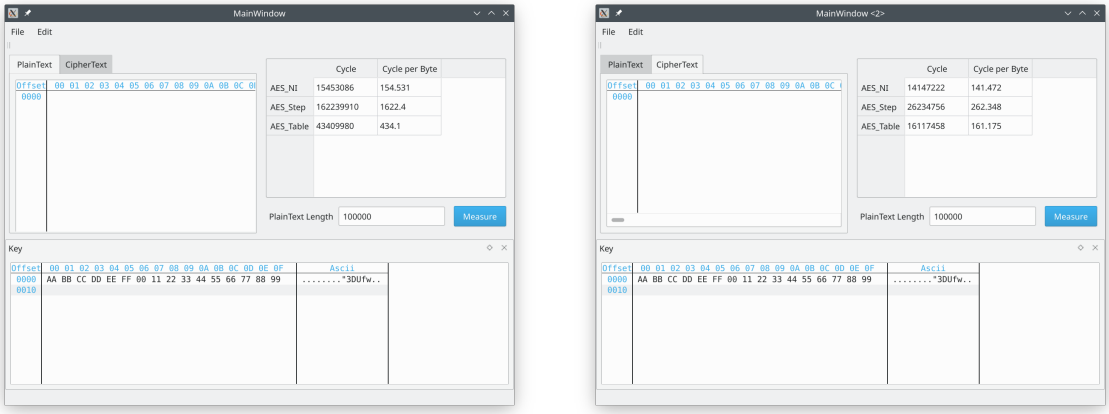
4 测试原理

给定密钥，设定明文长度，采用随机数发生器随机生成指定长度到明文。利用gcc提供的rdstc函数获取当前时间（精确到时钟周期），然后开始执行加密操作，加密完成后再次获取当前时间。求二者差值即可得到完成加密所需的总的时钟周期数。用该数据除以明文长度，即可到加密速度。

$$CyclePerBytes = \frac{Cycles}{PlainTextLength} = \frac{EndTime - StartTime}{PlainTextLength}$$

5 测试结果

在Debug和Release两种版本下，测试程序表现出了不同的行为：在Debug版本中，各种实现在效率上存在明显差距，AES_NI实现 > 查表实现 > 一般实现，而在Release版本中经过编译优化之后的各种实现的效率基本处于同一量级。



(a) Debug (b) Release

Figure 1: Test

如图1(a)所示，在Debug模式下，加密任意长度为100000 byte的随机字符串的测试中，AES_NI 实现的的效率为**154.531 cycles/byte**，效率约为一般实现AES_Step(**1622.4 cycles/byte**)的十倍。而查表法实现AES_Table的效率为**434.1 cycles/byte**，也显著高于一般实现。

在Release模式下的测试结果如图1(b)所示，在良好的编译优化下，三者之间的效率差距被大大缩小，但AES_NI与一般实现之间仍存在约2倍的差距。注意到AES_NI的效率并没有进一步提高，猜测可能是编译优化中对另外两种实现采用了向量化操作，从而将其效率提升到了与AES_NI相近的水平。

6 讨论

采用面向对象的设计，对算法进行了封装，但没有进一步测量对象构造的开销是否对加密效率存在显著影响，若需执行进一步更加严谨的测量，需要考虑到扣除这一部分的代价。