# 第二次作业

计15 宋驰 2021010797

## 1、AES-128-CTR

- 运行方式：

  ```
  g++ AES-CTR.cpp -o aes
  ```

  可以得到可执行文件aes

  ```
  ./aes e 1
  ```

  第一个参数用于指定是加密/解密/测速，e代表加密，d代表解密，t代表利用8K或者8M的数据进行测试运行速率

  第二个参数用于加密和解密时指定需要加解密的文件，有5个测例，因此该参数的范围是1-5，从而选择加解密哪份文件

  运行之后会将加解密的结果存在代码中指定的文件内

- 算法实现逻辑

  AES的加密是以块为单位，由于是CTR模式，因此只需要对ctr进行加密，然后跟明文的分块异或即可

  首先通过rcon数组完成密钥扩展。每次加密时，先将ctr计数器+1来获得需要加密的文字，然后进入加密的过程。先将128位的分组长度串转换成4*4的state，随后的操作均对state进行，直到最后一步后再从state转回128位串。随后按照AES加密的流程，依次完成S盒置换、行循环移位、列混淆、与密钥异或，共计10轮

  S盒置换通过提前设置好的sbox进行，行循环移位通过shift进行，列混淆通过提前设置的常量gf进行

  AES 的解密是加密的逆运算，由于是CTR模式，其实只需要把密文和加密后的ctr异或即可，因此和加密过程完全一致

- 优化细节
  - 使用inline函数，减少函数调用的开销
- 算法实现效率

  实现的加密算法和解密算法的效率在测试后可以达到100Mbps的标准

  - 下图是加密test_5.txt的效率，这次加密的数据大小达到了2048字节

    ```
    (base) songchi@DESKTOP-8P7N4AH:/mnt/d/CodeData/cryptography/lab2$ ./aes e 5
    2048
    数据 = 0.016384Mb
    时间 = 0.000155s
    效率 = 105.703Mbps
    ```

  - 下图是解密ciphertext_5.txt的效率，这次解密的数据大小达到了2048字节

    ```
    (base) songchi@DESKTOP-8P7N4AH:/mnt/d/CodeData/cryptography/lab2$ ./aes d 5
    2048
    数据 = 0.016384Mb
    时间 = 0.00014s
    效率 = 117.029Mbps
    ```

  - 下图是加密8Kbits大小的随机数据的效率

```
(base) songchi@DESKTOP-8P7N4AH:/mnt/d/CodeData/cryptography/lab2$ ./aes t 1
1000
数据 = 0.008Mb
时间 = 7.7e-05s
效率 = 103.896Mbps
```

- 下图是加密8Mbits大小的随机数据的效率

```
(base) songchi@DESKTOP-8P7N4AH:/mnt/d/CodeData/cryptography/lab2$ ./aes t 2
1000000
数据 = 8Mb
时间 = 0.076635s
效率 = 104.391Mbps
```

- HonorCode：

  常量数组的实现参考自https://github.com/Trinkle23897/Undergraduate/tree/master/%E7%8E%B0%E4%BB%A3%E5%AF%86%E7%A0%81%E5%AD%A6/AES%2BSHA

  输入串转128位hex串的方法和王豪达同学进行讨论

## 2、SHA-256

- 运行方式：

  ```
  g++ SHA-256.cpp -o sha256
  ```

  可以得到可执行文件sha256

  ```
  ./sha256 n 1
  ```

  第一个参数用于指定是正常哈希/测速，n代表正常哈希，t代表利用8K或者8M的数据进行测试运行速率

  第二个参数用于正常哈希时指定需要哈希的文件，有5个测例，因此该参数的范围是1-5，从而选择哈希哪份文件

  运行之后会将哈希的结果存在代码中指定的文件内

- 算法实现逻辑

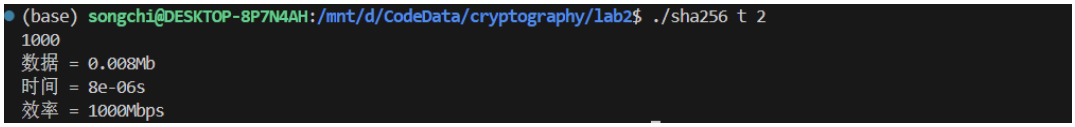
  首先初始化hash[0]-hash[7]，用于第一次hash中的a-h变量的值。随后对输入的串进行操作，每提取到256位就进行SHA-256的核心64轮运算，按下图更新各变量

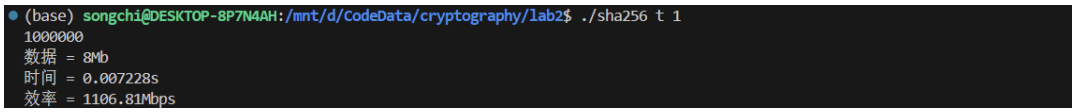
$$
\begin{aligned}
T_1 &\leftarrow h + \Sigma_1(e) + Ch(e,f,g) + K_j + W_j \\
T_2 &\leftarrow \Sigma_0(a) + M_{aj}(a,b,c) \\
h &\leftarrow g \\
g &\leftarrow f \\
f &\leftarrow e \\
e &\leftarrow d + T_1 \\
d &\leftarrow c \\
c &\leftarrow b \\
b &\leftarrow a \\
a &\leftarrow T_1 + T_2
\end{aligned}
$$

  利用这些中间变量计算每轮结束后的hash[0]-hash[7]，64轮结束后得到最后的hash值

- 优化细节
  - 将上面提到的函数通过宏定义的方式来实现，因为这些函数都比较简单，宏定义比较容易实现，减少函数调用的开销
- 算法实现效率

  实现的SHA-256算法的效率远超100Mbps的需求
  - 下图是哈希8Kbits大小的随机数据的效率

    
    
  - 下图是哈希8Mbits大小的随机数据的效率

    
    
- HonorCode：

  思路参考自https://zhuanlan.zhihu.com/p/94619052和https://github.com/ilvn/SHA256/blob/main/sha256.c

## 3、SHA3-256

- 运行方式：

  ```
  g++ SHA3.cpp -o sha3
  ```

  可以得到可执行文件sha256

  ```
  ./sha3 n 1
  ```

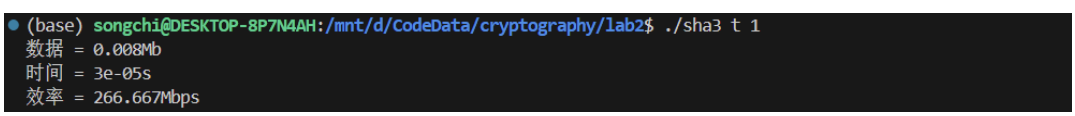
  第一个参数用于指定是正常哈希/测速，n代表正常哈希，t代表利用8K或者8M的数据进行测试运行速率

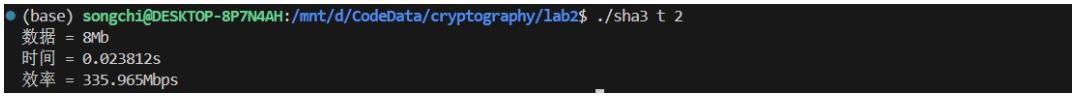
  第二个参数用于正常哈希时指定需要哈希的文件，有5个测例，因此该参数的范围是1-5，从而选择哈希哪份文件

  运行之后会将哈希的结果存在代码中指定的文件内
- 算法实现逻辑

  对输入的串处理和之前的算法一致，得到16进制表示的字符串，用于输入函数

  算法核心是sha3函数，通过Theta、Rho、Pi、Chi和Iota五个步骤来实现，Theta通过计算coefficients数组并对hash数组异或来实现线性变换，Rho和Pi对数据循环左移并排列，循环左移通过实现的rotateLeft来计算，Chi通过对hash数组的位运算和异或实现了非线性变换，Iota是最后的常量添加
- 优化细节
  - 将rotateLeft和swap的实现放在了宏定义来实现，减少函数调用的开销
  - 将用于接收输入的buffer数组开的较小，减小程序的开销
- 算法实现效率
  - 下图是哈希8Kbits大小的随机数据的效率

    
    
  - 下图是哈希8Mbits大小的随机数据的效率




- HonorCode：

  思路参考自张思源同学