

现代密码学 第二次大作业报告

黄昱炜 2018011368

1 运行环境

搭载 Ubuntu 20.04.2 LTS 系统的虚拟机，CPU 为 Intel Core i7-8750H。

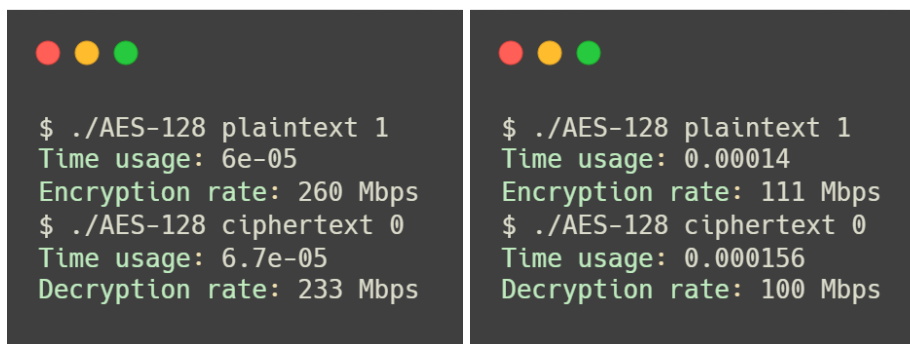
2 AES-128

2.1 优化方法

AES-128 加密算法的核心步骤有四个：ByteSub，ShiftRow，MixColumn，AddRoundKey。优化 ByteSub 可以考虑预先计算出 S 盒和 S 盒的逆，并以数组形式保存，计算过程中直接使用即可。ShiftRow 是简单的循环移位操作，没有很大的优化空间。MixColumn 涉及 $GF(2^8)$ 上的乘法，较为复杂，因此可以采用与 ByteSub 相同的方法，预先计算出乘法结果。AddRoundKey 是简单的异或操作，同样没有很大的优化空间。

另外，在实际的代码实现中，我还采用了诸如循环展开等一些简单的优化方法。

2.2 实验结果



Configuration	Time usage (plaintext)	Encryption rate	Time usage (ciphertext)	Decryption rate
Default (-Og)	6e-05	260 Mbps	6.7e-05	233 Mbps
Without optimization	0.00014	111 Mbps	0.000156	100 Mbps

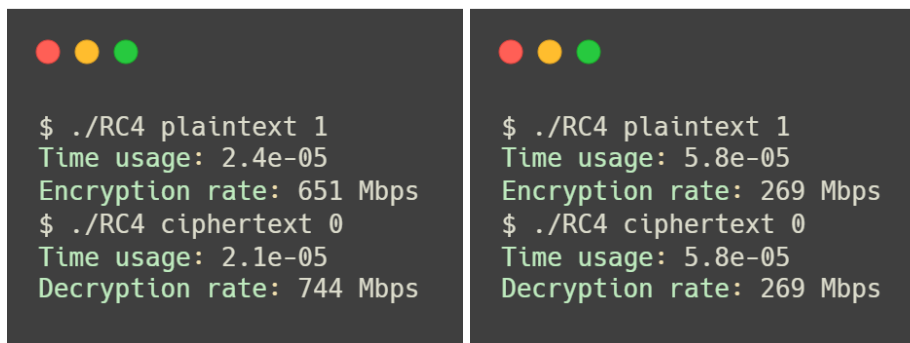
左图是使用默认-Og 优化选项时的性能表现，右图是不使用优化选项时的性能表现，可以看出都达到了 100Mbps 的要求。目录中提供了一对明密文 plaintext 和 ciphertext，可用于检查结果的正确性。更多有关编译、运行、检查的说明请见 readme 文档。

3 RC4

3.1 优化方法

RC4 算法本身较为简单：先进行 key-scheduling，然后产生流密码和明文 (密文) 异或，即可完成加密 (解密)。因此在实现过程中，我没有对该算法进行针对性优化。

3.2 实验结果



The image shows two side-by-side terminal windows. The left window shows the performance of the RC4 program with the -Og optimization flag. The right window shows the performance without optimization. Both windows display the same commands: './RC4 plaintext 1' and './RC4 ciphertext 0'. The left window shows significantly higher encryption and decryption rates (651 Mbps and 744 Mbps respectively) compared to the right window (269 Mbps for both).

Command	Time usage	Rate
./RC4 plaintext 1	2.4e-05	651 Mbps
./RC4 ciphertext 0	2.1e-05	744 Mbps

Command	Time usage	Rate
./RC4 plaintext 1	5.8e-05	269 Mbps
./RC4 ciphertext 0	5.8e-05	269 Mbps

同样，左图是使用默认-Og 优化选项时的性能表现，右图是不使用优化选项时的性能表现，可以看出都达到了 100Mbps 的要求。目录中提供了一对明密文 plaintext, ciphertext 以及初始密钥 key，可用于检查结果的正确性。更多有关编译、运行、检查的说明请见 readme 文档。

4 SHA-3-256

4.1 优化方法

SHA-3 系列算法的核心步骤是 KECCAK 海绵函数，其中的每一轮包含 $\theta, \rho, \pi, \chi, \iota$ 五个子函数。除去普通的循环展开外，我重点对 ρ 函数和 π 函数做了优化。观察 ρ 函数的迭代步

3. For t from 0 to 23:
 - a. for all z such that $0 \leq z < w$, let $A'[x, y, z] = A[x, y, (z - (t+1)(t+2)/2) \bmod w]$;
 - b. let $(x, y) = (y, (2x+3y) \bmod 5)$.
4. Return A' .

以及 π 函数的坐标变换步

1. For all triples (x, y, z) such that $0 \leq x < 5$, $0 \leq y < 5$, and $0 \leq z < w$, let $A'[x, y, z] = A[(x+3y) \bmod 5, x, z]$.
2. Return A' .

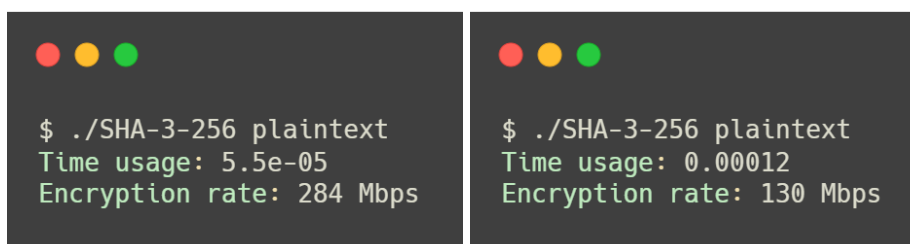
观察发现，在模 5 意义下， $(x, y) \rightarrow (y, 2x+3y)$ 的逆映射恰好是 $(x, y) \rightarrow (x+3y, x)$ ，因此我们可以将 π 函数结合在 ρ 函数里实现，在 ρ 函数迭代过程中同时完成 π 函数的坐标变换。

另外，由于 t 的迭代过程确定， (x, y) 的初值确定，我们可以预先计算出它们在变化过程中的值，需要时直接使用即可。类似处理的还有 ι 函数，观察其计算过程：

1. For all triples (x, y, z) such that $0 \leq x < 5$, $0 \leq y < 5$, and $0 \leq z < w$, let $A'[x, y, z] = A[x, y, z]$.
2. Let $RC = 0^w$.
3. For j from 0 to ℓ , let $RC[2^j - 1] = rc(j + 7i_r)$.
4. For all z such that $0 \leq z < w$, let $A'[0, 0, z] = A'[0, 0, z] \oplus RC[z]$.
5. Return A' .

图中 i_r 的迭代过程确定，因此 rc 函数的结果乃至最终 RC 的值都可以预先计算出来，这样 ι 函数就转化为一步简单的异或操作（状态数组 A 和预先计算出的 RC 进行异或）。

4.2 实验结果



同样，左图是使用默认-Og 优化选项时的性能表现，右图是不使用优化选项时的性能表现，可以看出都达到了 100Mbps 的要求。目录中提供了一份明文 plaintext 和对应的散列值 hashtext，可用于检查结果的正确性。更多有关编译、运行、检查的说明请见 readme 文档。