密码学实验报告 实验六

2019 年 4 月 17 日

1 AES 软件级快速实现

1.1 算法原理

AES 包含四个对字节矩阵的基本操作：字节替换、行移位、列混淆、轮密钥加。这些操作往往会极大地耗费计算资源，若以字节运算为单位，字节替换需要 16 次运算，行移位需要 12 次运算，列混淆需要 60 次甚至更多的运算，轮密钥加需要 16 次运算，每轮总共需要至少 102 次运算，这是 AES 运算效率低的根本原因所在。

然而，在软件级实现中，我们往往可以将三个不同的运算统一到一个查找表中：字节替换（非线性），行移位（线性），列混淆（线性），这样一来每轮的运算只需要访问查找表和轮密钥加，可以降低每轮需要的运算次数，使算法的效率得到较大的提升。

在 AES 算法中数据被描述为一个状态数组，形式如下：

$$\begin{matrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{matrix}$$

前向查找表算法的关键在于 T-tables 或 T-boxes 的实现，该表专门针对列混淆操作进行优化。列混淆利用有限域上的加法和乘法对输入状态的每一列字节实施变换，一般地，列混淆输入一列字节c_0, c_1, c_2, c_3，输出一列字节s_0, s_1, s_2, s_3：

$$s_0 = 2 * c_0 \oplus 3 * c_1 \oplus 1 * c_2 \oplus 1 * c_3$$

$$s_1 = 1 * c_0 \oplus 2 * c_1 \oplus 3 * c_2 \oplus 1 * c_3$$

$$s_2 = 1 * c_0 \oplus 1 * c_1 \oplus 2 * c_2 \oplus 3 * c_3$$

$$s_3 = 3 * c_0 \oplus 1 * c_1 \oplus 1 * c_2 \oplus 2 * c_3$$

记|符号为两个字节的拼接操作，基于有限域上加法和乘法的性质，上述运算步骤可以改写为以下形式：

$$s = s_0|s_1|s_2|s_3$$

$$s = 2 * c_0 \oplus 3 * c_1 \oplus 1 * c_2 \oplus 1 * c_3|$$
$$1 * c_0 \oplus 2 * c_1 \oplus 3 * c_2 \oplus 1 * c_3|$$
$$1 * c_0 \oplus 1 * c_1 \oplus 2 * c_2 \oplus 3 * c_3|$$
$$3 * c_0 \oplus 1 * c_1 \oplus 1 * c_2 \oplus 2 * c_3$$
$$s = (2 * c_0|1 * c_0|1 * c_0|3 * c_0) \oplus$$
$$(3 * c_1|2 * c_1|1 * c_1|1 * c_1) \oplus$$
$$(1 * c_2|3 * c_2|2 * c_2|1 * c_2) \oplus$$
$$(1 * c_3|1 * c_3|3 * c_3|2 * c_3)$$

由此我们可以看出，利用查找表来存储 32 位的列混淆输出是可行的，每个值由输入的字节来确定，由于每个字节为 8 位，所以查找表的元素个数为$2^8 = 256$，总大小为$256 * 32 = 8192$位。而对于来自原输入中每一列的运算是不同的，如此我们可以计算四个查找表te_0, te_1, te_2, te_3：

$$te_0[i] = 2 * i|1 * i|1 * i|3 * i$$
$$te_1[i] = 3 * i|2 * i|1 * i|1 * i$$
$$te_2[i] = 1 * i|3 * i|2 * i|3 * i$$
$$te_3[i] = 1 * i|1 * i|3 * i|2 * i$$
$$s = te_0[c_0] \oplus te_1[c_1] \oplus te_2[c_2] \oplus te_3[c_3]$$

不仅如此，我们还可以进一步对每轮的操作进行优化，将字节替换操作也统一到前向查找表中，并在计算中直接使用行移位，这里用到了 Rijndael 算法中行移位和字节替换可以交换的性质。

$$te_0[i] = 2 * S[i]|1 * S[i]|1 * S[i]|3 * S[i]$$
$$te_1[i] = 3 * S[i]|2 * S[i]|1 * S[i]|1 * S[i]$$
$$te_2[i] = 1 * S[i]|3 * S[i]|2 * S[i]|3 * S[i]$$
$$te_3[i] = 1 * S[i]|1 * S[i]|3 * S[i]|2 * S[i]$$
$$R_0 = te_0[b_0] \oplus te_1[b_5] \oplus te_2[b_{10}] \oplus te_3[b_{15}]$$
$$R_1 = te_0[b_4] \oplus te_1[b_9] \oplus te_2[b_{14}] \oplus te_3[b_3]$$
$$R_2 = te_0[b_8] \oplus te_1[b_{13}] \oplus te_2[b_2] \oplus te_3[b_7]$$
$$R_3 = te_0[b_{12}] \oplus te_1[b_1] \oplus te_2[b_6] \oplus te_3[b_{11}]$$

随后对每一行再与轮密钥相加，即可得到下一轮的输入状态，最后一轮中没

有列混淆，所以我们只需进行字节替换和行移位操作，最终输出理想的密文。

在解密过程中，只需要将加密过程的每个步骤取逆，根据我们在 AES 算法中的推导，每一轮中的运算顺序可以与加密时一致，只是需要将除了第一个和最后一个的所有密钥进行逆向列混淆操作。

同理，我们可以推导出以下式子：

$$s_0 = 14 * c_0 \oplus 11 * c_1 \oplus 13 * c_2 \oplus 9 * c_3$$

$$s_1 = 9 * c_0 \oplus 14 * c_1 \oplus 11 * c_2 \oplus 13 * c_3$$

$$s_2 = 13 * c_0 \oplus 9 * c_1 \oplus 14 * c_2 \oplus 11 * c_3$$

$$s_3 = 11 * c_0 \oplus 13 * c_1 \oplus 9 * c_2 \oplus 14 * c_3$$

$$s = s_0|s_1|s_2|s_3$$

$$\begin{aligned} s = {} & 14 * c_0 \oplus 11 * c_1 \oplus 13 * c_2 \oplus 9 * c_3| \\ & 9 * c_0 \oplus 14 * c_1 \oplus 11 * c_2 \oplus 13 * c_3| \\ & 13 * c_0 \oplus 9 * c_1 \oplus 14 * c_2 \oplus 11 * c_3| \\ & 11 * c_0 \oplus 13 * c_1 \oplus 9 * c_2 \oplus 14 * c_3 \end{aligned}$$

$$\begin{aligned} s = {} & (14 * c_0|9 * c_0|13 * c_0|11 * c_0) \oplus \\ & (11 * c_1|14 * c_1|9 * c_1|13 * c_1) \oplus \\ & (13 * c_2|11 * c_2|14 * c_2|9 * c_2) \oplus \\ & (9 * c_3|13 * c_3|11 * c_3|14 * c_3) \end{aligned}$$

使用与加密过程同样的简化步骤，同样我们可以创建 4 个 T-table 实现解密的查找表

$$td_0[i] = 14 * IS[i]|9 * IS[i]|13 * IS[i]|11 * IS[i]$$

$$td_1[i] = 11 * IS[i]|14 * IS[i]|9 * IS[i]|14 * IS[i]$$

$$td_2[i] = 13 * IS[i]|11 * IS[i]|14 * IS[i]|9 * IS[i]$$

$$td_3[i] = 9 * IS[i]|13 * IS[i]|11 * IS[i]|14 * IS[i]$$

$$R_0 = td_0[b_0] \oplus td_1[b_{13}] \oplus td_2[b_{10}] \oplus td_3[b_7]$$

$$R_1 = td_0[b_4] \oplus td_1[b_1] \oplus td_2[b_{14}] \oplus td_3[b_{11}]$$

$$R_2 = td_0[b_8] \oplus td_1[b_5] \oplus td_2[b_2] \oplus td_3[b_{15}]$$

$$R_3 = td_0[b_{12}] \oplus td_1[b_9] \oplus td_2[b_6] \oplus td_3[b_3]$$

随后将每一行与逆向轮密钥相加，得到下一轮的输入状态，最后一轮中没有

列混淆，所以我们只需进行字节替换和行移位操作，最终输出对应的明文。

接下来可以对查表法进行简单的分析，其空间占用为：

$$(4\ \ bytes) * (256\ \ entries) * (4\ \ tables) = 4\ \ kbytes$$

每轮进行的运算次数为：

$$\big((4\ \ look\text{-}up) + (3\ \ xors)\big) * (4\ \ columns) + (4\ \ xors\ \ with\ \ keys) = 32\ \ ops/r$$

相比之前的 102 次每轮，已经得到了极大的优化。

1.2 算法实现的伪代码

AES 快速加密算法 利用快速查表法进行 AES 加密

输入： 128 位明文P，128 位密钥K
输出： 128 位密文C

function $aes\_encrypt(K, P)$
    $k \leftarrow ExpandKey(K)$
    $s \leftarrow P$
    for $j \leftarrow 0\ \ to\ \ 3$ **do**
        $s_{4*j:4*(j+1)} \leftarrow s_{4*j:4*(j+1)} \oplus k[j]$
    end for
    for $i \leftarrow 1\ \ to\ \ 9$ **do**
        $r_0 \leftarrow te_0[s_0] \oplus te_1[s_5] \oplus te_2[s_{10}] \oplus te_3[s_{15}] \oplus k[4*i]$
        $r_1 \leftarrow te_0[s_4] \oplus te_1[s_9] \oplus te_2[s_{14}] \oplus te_3[s_3] \oplus k[4*i+1]$
        $r_2 \leftarrow te_0[s_8] \oplus te_1[s_{13}] \oplus te_2[s_2] \oplus te_3[s_7] \oplus k[4*i+2]$
        $r_3 \leftarrow te_0[s_{12}] \oplus te_1[s_1] \oplus te_2[s_6] \oplus te_3[s_{11}] \oplus k[4*i+3]$
        $s \leftarrow r$
    end for
    $r_0 \leftarrow (sbox[s_0]||sbox[s_5]||sbox[s_{10}]||sbox[s_{15}]) \oplus k[40]$
    $r_1 \leftarrow (sbox[s_4]||sbox[s_9]||sbox[s_{14}]||sbox[s_3]) \oplus k[41]$
    $r_2 \leftarrow (sbox[s_8]||sbox[s_{13}]||sbox[s_2]||sbox[s_7]) \oplus k[42]$
    $r_3 \leftarrow (sbox[s_{12}]||sbox[s_1]||sbox[s_6]||sbox[s_{11}]) \oplus k[43]$
    $C \leftarrow r_1|r_2|r_3|r_4$
    return C
end function

AES 快速解密算法 利用快速查表法进行 AES 解密

输入： 128 位密文C，128 位密钥K
输出： 128 位明文P

function $aes\_decrypt(K, C)$
    $k \leftarrow ExpandKey(K)$
    $s \leftarrow C$
    for $j \leftarrow 0\ \ to\ \ 3$ **do**

$s_{4*j:4*(j+1)} \leftarrow s_{4*j:4*(j+1)} \oplus ik[40+j]$

end for

for $i \leftarrow 9$ to 1 **do**

$r_0 \leftarrow td_0[s_0] \oplus td_1[s_{13}] \oplus td_2[s_{10}] \oplus td_3[s_7] \oplus ik[4*i]$

$r_1 \leftarrow td_0[s_4] \oplus td_1[s_1] \oplus td_2[s_{14}] \oplus td_3[s_{11}] \oplus ik[4*i+1]$

$r_2 \leftarrow td_0[s_8] \oplus td_1[s_5] \oplus td_2[s_2] \oplus td_3[s_{15}] \oplus ik[4*i+2]$

$r_3 \leftarrow td_0[s_{12}] \oplus td_1[s_9] \oplus td_2[s_6] \oplus td_3[s_3] \oplus ik[4*i+3]$

$s \leftarrow r$

end for

$r_0 \leftarrow (isbox[s_0]||isbox[s_{13}]||isbox[s_{10}]||isbox[s_7]) \oplus ik[0]$

$r_1 \leftarrow (isbox[s_4]||isbox[s_1]||isbox[s_{14}]||isbox[s_{11}]) \oplus ik[1]$

$r_2 \leftarrow (isbox[s_8]||isbox[s_5]||isbox[s_2]||isbox[s_{15}]) \oplus ik[2]$

$r_3 \leftarrow (isbox[s_{12}]||isbox[s_9]||isbox[s_6]||isbox[s_3]) \oplus ik[3]$

$P \leftarrow r_1|r_2|r_3|r_4$

return P

end function

1.3 并行化加解密算法实现

通过对分组加密模式的进一步观察，我发现在 ECB 模式中可以利用多进程来进一步加速 AES 算法，可以在多核处理器上有着更佳的表现。电码本（ECB）模式一次处理一组明文分块，每次使用相同的密钥加密。明文若长于 b 位，则可简单将其分为 b 位一组的块，有必要则可对最后一块进行填充。解密也是一次执行一块，且使用相同的密钥。基于以上的加密模式，由于每次加解密使用的密钥相同，我们可以将不同组的加解密分配到不同的处理器核心上运行，具体实现使用了 Python 的多进程库 multiprocessing，将需要加密的数据块分为 n 组，其中 n 为处理器核心数目，这样做是为了减少进程创建的开销，之后再并行地计算这些数据分组，将算法的输出合并即可得到理想的结果。

然而此方法也有一定的限制，例如在 CBC 和 CFB 等模式中，算法当前的加解密需要依赖前一个分组的输出，就不能使用当前的并行化处理，但是经过进一步观察，CBC 模式和 CFB 模式解密算法中的部分运算步骤仍可以并行化，此处没有做具体的实验分析。

1.4 测试样例及运行结果

为了充分对算法的性能进行验证，我们对于优化前、优化后、优化后且使用

了并行处理的 AES 分组加密算法进行了实验验证和对比分析，实验环境如表 1 所示。

表 1 实验环境

处理器	RAM	工作模式	编程语言
Intel i7-6700HQ	8GB	ECB	Python 3.6

接下来输入不同大小的文件并记录算法的执行时间，结果如表 2 所示。

表 2 算法运行时间（单位：秒）

文件大小	普通实现	快速实现	快速+并行
100KB	41.242	0.214	0.506
200KB	81.796	0.435	0.618
500KB	206.725	1.052	0.765
1MB	-	2.155	1.128
3MB	-	6.573	2.289
5MB	-	10.797	3.431
10MB	-	22.183	6.637
20MB	-	42.774	12.847
50MB	-	106.371	30.764
初始化时间：0.398			

利用 Python 的 matplotlib 模块绘制数据曲线，结果如图 1 所示。

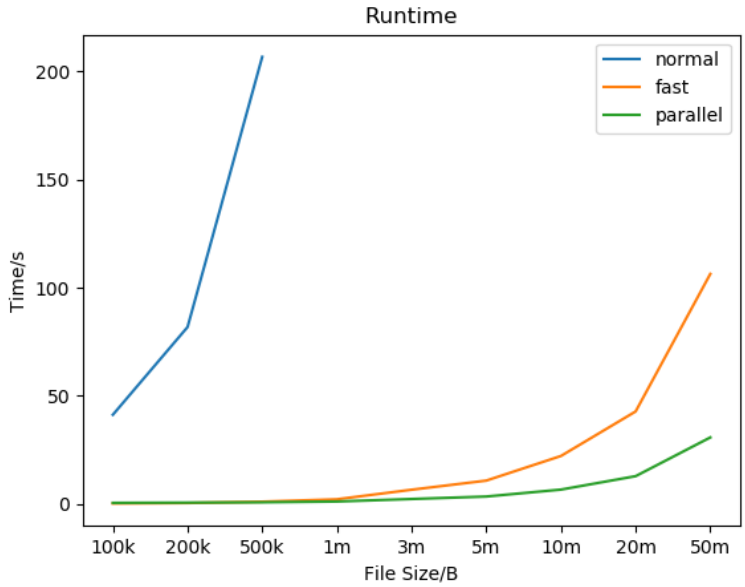

图 1 算法运行时间对比

从表中数据和图中曲线可以看出，快速查表法平均比普通实现方法提高了 200 倍的速度，而对于较大的文件，并行化可以提高 3 倍的速度，最终我们在当前的实验环境下达到了 1.63MB/s 的加密速度。

除此之外，为了验证并行化处理时的快速实现是否可以正确地进行加解密运

算，我们挑选了一张图片对其进行了加解密操作，原文件为 1m.png，加密后的文件为 en_1m.png，解密后的文件为 de_1m.png，结果如图 2 所示。

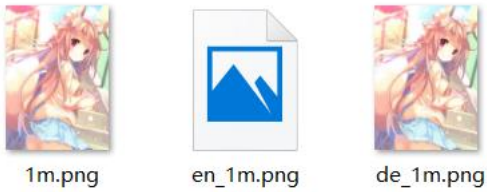

图 2 对文件的加解密验证

2 总结

对于 AES 的软件级快速实现，我们选择了在多种平台上都可以实现的查表法，首先思考原来实现方式中对计算效率影响较大的步骤，然后对于 AES 每轮中的步骤进行详细分析，发现可以将字节替换、行移位、列混淆操作统一到一个前向查找表中，如此便可以极大地缩减每轮中必要的运算步骤，从而提高 AES 算法的效率，解密步骤中同理也可以进行类似加密时的优化。除此之外，我们考虑了一种并行化的分组加密方式，在我们的实验环境下可以达到比优化后再提高三倍的加速效果。最终我们得到的算法可以达到 1.63MB/s 的加解密速度，较原先的实现方式提高了约 600 倍的速度，我们的思路在当前的实验环境下得到了充分验证。然而算法的速度也因为 Python 本身的执行效率而被限制，在之后的工作中，我们可以考虑在效率更快的语言，例如 C++、Java 等复现该代码，以达到工程级别上的速度标准。