**201722210013-苏文元-实验2-V2**

1. **实验目的**

熟悉使用Linux常用的操作命令，安装虚拟机。

1. **实验内容**

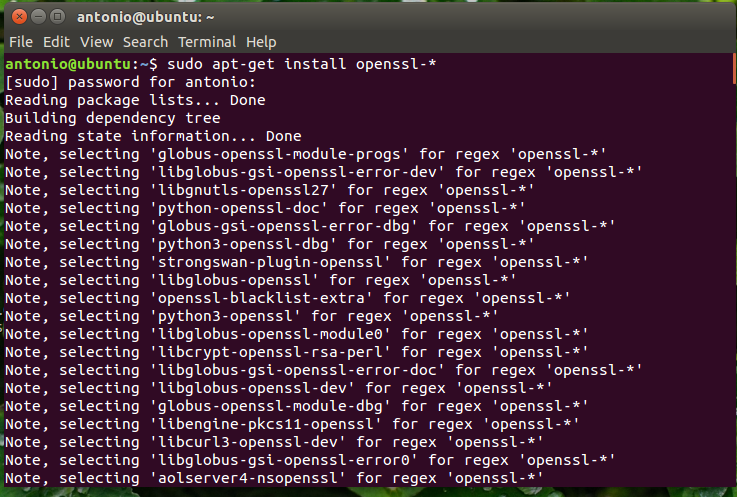
1. 安装OpenSSL；

2. 使用OpenSSL进行DES/AES加密、解密；

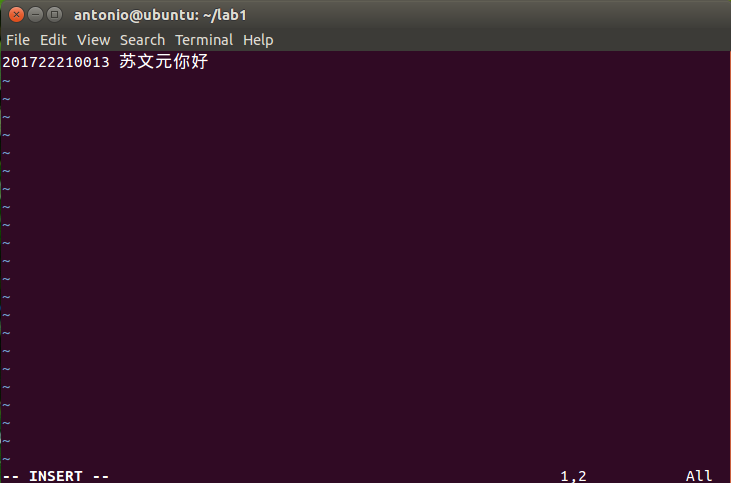
3. 比较对称加密算法的效率；

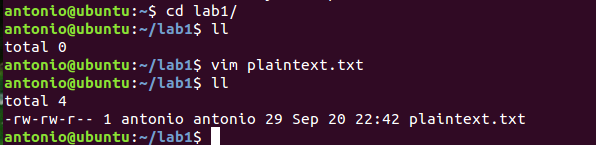
4. 采用计算机语言编程实现DES算法的加密与解密过程。

1. **实验过程**
2. 安装openssl；



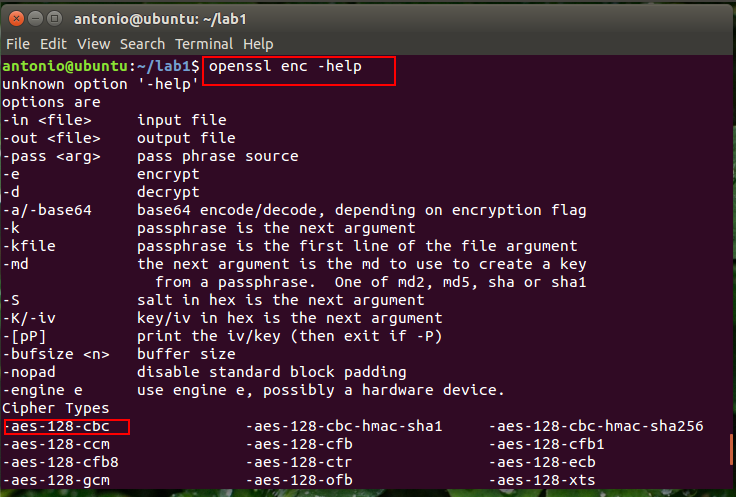
1. 使用OpenSSL进行DES/AES加密、解密
2. 创建明文*plaintext.txt*，内容为自己的学号+任意一段话。



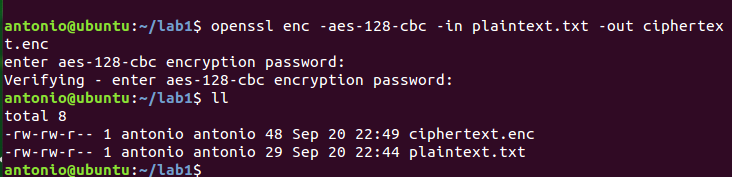


1. 查阅OpenSSL命令的使用方法，对明文做加密，密文*ciphertext.enc*。

提示：*openssl enc -aes-128-cbc* –*in* 明文–*out* 密文

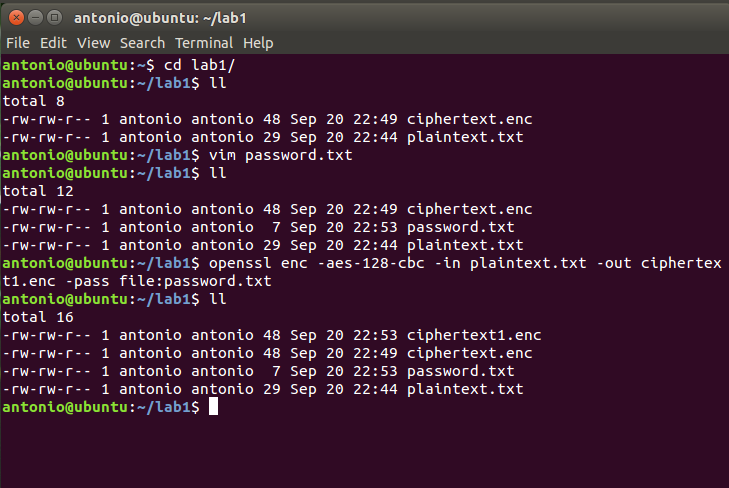


密码设置为123456

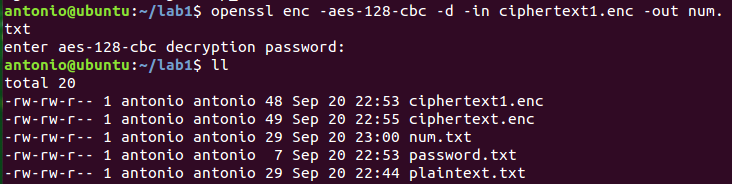


1. 如果密码保存在文本文件*password.txt*中，该如何用*password.txt*对明文做加密呢？

openssl enc -aes-128-cbc -in plain.txt -out out.txt -pass file:password.txt

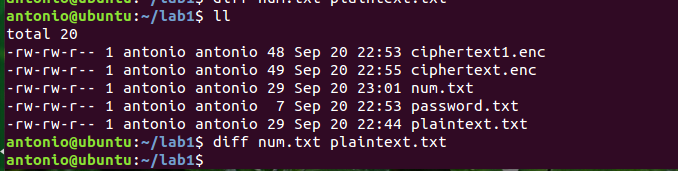
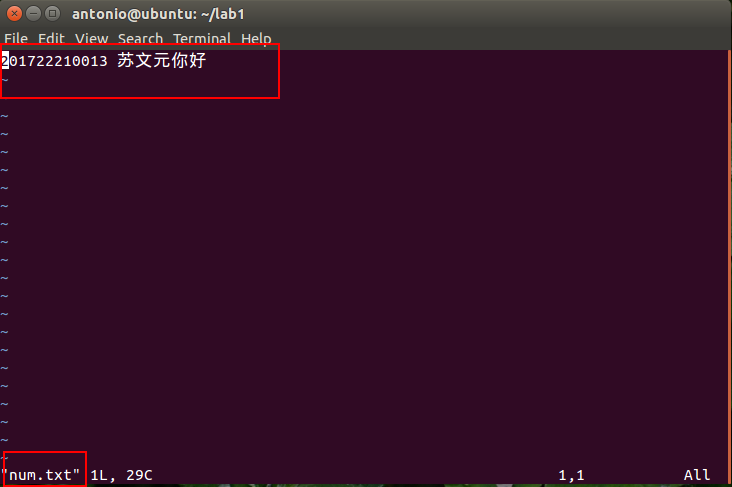


1. 查阅OpenSSL命令的使用方法，对密文做解密，解密后文件为*decrypt.txt*。



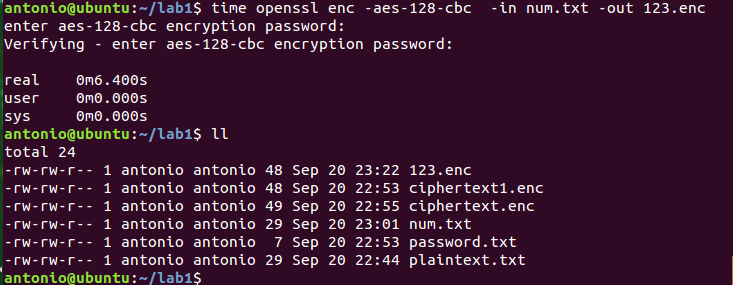
1. 解密的文件是什么？如何比较*plaintext.txt*与*decrypt.txt*两个文件之间的差异？

提示：*diff*



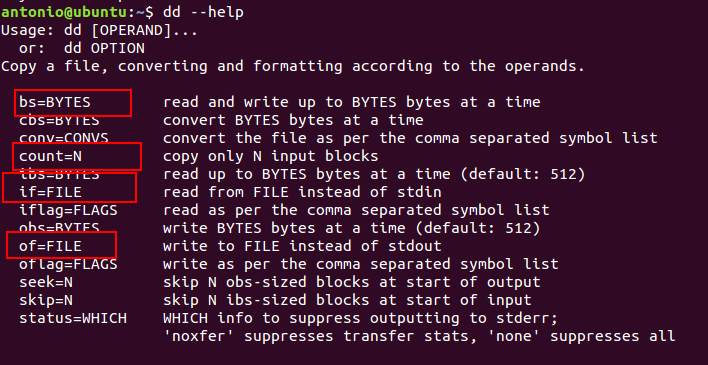
1. AES加密的效率
2. 如何准确得到AES加密过程所花费的时间？第2步的加密过程用了多久？

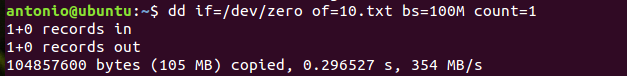
提示：*time*



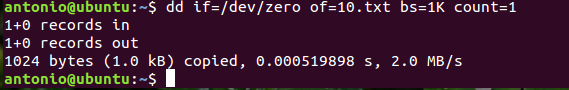
1. 创建一个较大的明文，文件大小为100M，执行多次AES加密过程，记录所花费的平均时间。

提示：*dd*





1. 创建一个较小的明文，文件大小为1byte，执行多次AES加密过程，记录所花费的平均时间。

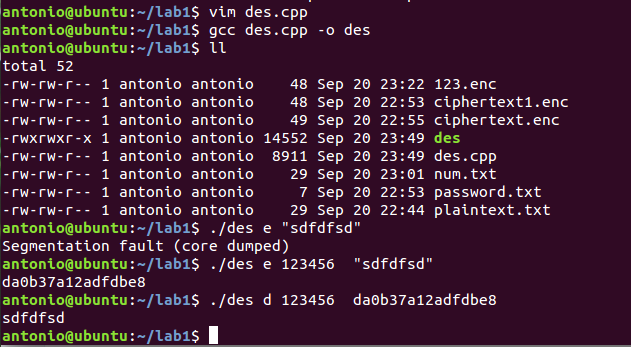


1. 明文大小对加密时间有影响吗？

由2）和3）知文件大小对时间有影响。

1. 实现DES算法

|  |
| --- |
| #include <stdio.h>  #include <string.h>    #define maxn 0x8000 // 理论支持明文长度  #define ENCODE 0,16,1 // 加密用的宏  #define DECODE 15,-1,-1 // 解密用的宏    // 明文初始置换  char msg\_ch[64] = {  58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,  62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,  57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,  61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7  };    // 密钥初始置换  char key\_ch[56] = {  57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,  10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,  63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,  14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4  };    // 扩展置换  char msg\_ex[48] = {  32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,  8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17,  16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,  24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1  };    // 每轮密钥的位移  char key\_mov[16] = {  1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1  };    // 压缩置换  char key\_cmprs[48] = {  14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,  23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,  41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,  44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32  };    // S 盒置换  char s\_box[8][4][16] = {  // S1  14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,  0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,  4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,  15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13,  // S2  15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,  3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,  0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,  13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9,  // S3  10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,  13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,  13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,  1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12,  // S4  7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,  13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,  10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,  3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14,  // S5  2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,  14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,  4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,  11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3,  // S6  12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,  10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,  9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,  4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13,  // S7  4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,  13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,  1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,  6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12,  // S8  13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,  1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,  7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,  2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11  };    // P 盒置换  char p\_box[32] = {  16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,  2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25  };    // 末置换  char last\_ch[64] = {  40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,  38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,  36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,  34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25  };    // hash 置换，将加密后的密文置换为可读明文  char hs\_ch[20] = "0123456789abcdef";  char sh\_ch[128];  void init\_trans() {  char i;  for (i = 0; i < 16; i++)  sh\_ch[hs\_ch[i]] = i; // 完成hash转换的对应  }    char msg[maxn], key[16], res[32];  char msgb[72], msgbt[72], keyb[18][72];    // 字符转成二进制  void ChToBit(char\* dest, char\* src, int length) {  int i, j;  char t;  for (i = 0; i < length; i++) {  for (j = 8, t = src[i]; j > 0; j--) {  dest[(i << 3) + j] = t & 1; // 取字符末位  t >>= 1;  }  }  }    // 二进制转成字符  void BitToCh(char\* dest, char\* src, int length) {  int i;  for (i = 0; i < length << 3; i++) {  dest[i >> 3] <<= 1;  dest[i >> 3] |= src[i + 1]; // 添加到末位  }  dest[length] = 0;  }    // 批置换，以offset为偏移，以count为长度  void BatchSet(char\* dest, char\* src, char\* offset, int count) {  int i;  for (i = 0; i < count; i++)  dest[i + 1] = src[offset[i]];  }    // 得到16轮所需的密钥  void getKeys() {  char tk[128], bk[72];  char\* ptk = tk;  int i, j;  for (i = 0; i < 8; i++)  key[i] <<= 1; // 跳过奇偶校验位  ChToBit(bk, key, 8);  BatchSet(tk, bk, key\_ch, 56);  for (i = 0; i < 16; i++) {  for (j = 0; j < key\_mov[i]; j++, ptk++) {  ptk[57] = ptk[28];  ptk[28] = ptk[1];  // ptk++ 为亮点所在，实质上每一位都没有左移，只是指针右移了  }  BatchSet(keyb[i], ptk, key\_cmprs, 48);  }  }    // 将加密后的密文转换为可读的明文  void msgPro(char\* dest, char\* src) {  int i, j;  for (i = 0; i < 16; i++) {  dest[i] = 0;  for (j = 1; j <= 4; j++) // 取4位按hash构造一个字符  dest[i] = (dest[i] << 1) | src[i \* 4 + j];  dest[i] = hs\_ch[dest[i]];  }  dest[i] = 0;  }    // 将密文转换为真正的密文  void dropMsg(char\* dest, char\* src) {  int i;  for (i = 0; i < 16; i++) { // 为hash的逆运算过程  dest[i >> 1] = (dest[i >> 1] << 4) | sh\_ch[src[i]];  }  }    // 通用加/解密函数，后面三个参数由宏ENCODE和DECODE提供  void DES(char\* pmsg, int st, int cl, int step) {  int i, row, col;  char r[64], rt[48], s[8];  ChToBit(msgbt, pmsg, 8);  BatchSet(msgb, msgbt, msg\_ch, 64); // 初始置换  for (; st != cl; st += step) {  memcpy(rt, msgb + 33, 32);  BatchSet(r, msgb + 32, msg\_ex, 48); // 扩展置换  for (i = 1; i <= 48; i++)  r[i] ^= keyb[st][i]; // 异或操作  // s\_box 代替  for (i = 0; i < 48; i += 6) {  row = col = 0;  row = r[i + 1] << 1 | r[i + 6];  col = (r[i + 2] << 3) | (r[i + 3] << 2) | (r[i + 4] << 1) | r[i + 5];  s[i / 12] = (s[i / 12] <<= 4) | s\_box[i / 6][row][col];  }  ChToBit(r, s, 4);  BatchSet(msgb + 32, r, p\_box, 32); // p\_box 置换  for (i = 1; i <= 32; i++)  msgb[i + 32] ^= msgb[i]; // 异或  memcpy(msgb + 1, rt, 32);  }  memcpy(msgbt + 33, msgb + 1, 32);  memcpy(msgbt + 1, msgb + 33, 32);  BatchSet(msgb, msgbt, last\_ch, 64); // 末置换  if (step == 1)  msgPro(res, msgb); // 使密文可读  else  BitToCh(res, msgb, 8); // 转为原明文  }    /\* 本程序为其他程序调用，所有参数自己定义  \* 详细参数：  \* des d/e key msg  \* d——解密 e——加密  \* key——密钥  \* msg——要加/解密的明/密文  \*/  int main(int arg, char\* arv[]) {  if (arg < 3) {  printf("Input Error");  return 0;  }  init\_trans();    // 读取参数过程  char mode = arv[1][0];  strcpy(key, arv[2]);  strcpy(msg, arv[3]);    getKeys(); // 得到16轮要用到的密钥    int i;  if (mode == 'e') {  for (i = 0; msg[i]; i += 8) {  DES(msg + i, ENCODE); // 加密  printf("%s", res);  }  } else if (mode == 'd') {  for (i = 0; msg[i]; i += 16) {  dropMsg(res, msg + i); // 将密文转换为真正的密文  DES(res, DECODE); // 解密  printf("%s", res);  }  } else {  printf("Input Error!!!");  }  printf("\n");  return 0;  } |



1. **实验结论**

成功完成试验内容。