**模运算（Modular Arithmetic）**

同余（Congruence）:

若两个整数a和b的差是m的倍数，则称它们在模m下同余。即存在一个整数k，使得： a−b=k⋅m这表示为： a≡b (mod m)

**基本性质：**

自反性, 对称性,传递性,兼容性

**ssh -p 8888 admin2700@localhost**

使用SSH连接到本地的一个Ubuntu服务器。

用户在命令行输入了连接命令

用户试图通过端口8888连接到本地主机上的用户“admin2700”。

(通过SSH远程登录到一台运行Ubuntu的服务器)

**cmd指令：**

cd 切换资料夹位置

cd\ 切换到其根目录

cd.. 退回到上一层

cd ~ 返回用户主目录 ~ 这是用户主目录home directory的快捷方式

ls ~ 列出主目录的文件

ls 列出当前目录的内容

pwd显示当前工作目录的路径

cp 复制文件 cp source\_file destination\_file

mv 移动或重命名文件 mv source\_file destination\_file

rm 删除文件 rm file

mkdir 创建目录 mkdir directory\_name

cat file 显示文件内容 cat welcome.txt

ls -| 详细列出文件和根目录的属性

ls/ 列出根目录的内容

cd-： 返回上一个工作目录

cat：显示文件内容

./: 运行 ./hello

echo "This is a new file." > newfile.txt:创建一个简单的文本文件

sudo：它允许用户以另一用户（通常是超级用户或 root 用户）的权限运行程序。

**LAB1**

* connecting to the Virtual Machine
  + connecting
    - ssh -p PORT USER@HOSTNAME
    - ssh -p 8888 admin2700@localhost
  + disconnecting
    - “Send the shutdown signal”
* transferring files
  + sftp -P PORT USER@HOSTNAME
    - 在虚拟机（VM）和本地机器之间传输文件，可以使用安全文件传输协议（SFTP）
  + sftp command
    - sftp> ls 查看远程服务器当前目录的内容
    - sftp> get remote\_file 下载文件
    - sftp> put local\_file 上传文件
    - sftp> cd directory\_name 更改远程服务器目录
    - sftp> lcd local\_directory 更改本地目录
    - To exit sftp, use the command “bye”.
    - sftp> put newfile.txt 上传 newfile.txt 文件到 Lab VM
* switching users
  + su -l alice: login as alice 小写L
  + exit ：返回 admin2700
* files and directories
  + user directories are located under the folder/home.
  + pwd: print working directory,writes the full pathname (/home/alice)
  + ls[-options] [regular expression of filenames/directories] :list files in the specified working directory
    - . : 表示当前目录
    - .. : 表示当前目录的父目录
    - ~ ：表示当前用户的主目录
    - ls -a：列出所有文件，包括隐藏文件
    - ls -l： 以长格式显示文件详细信息
    - ls -lh：以人类可读格式显示文件大小
    - Is .. :列出当前目录的父目录中的所有文件
    - ls ~: 列出用户主目录中的所有文件
    - ls -la ~:以长格式显示用户主目录中的所有文件，包括隐藏文件
  + "wildcard" expressions “globbing”
    - asterisk character "\*" 匹配0个或多个字符
      * ls\* :list all the files and directories(it does not however list the hidden files/directories (whose names start with a dot))
      * ls -al /bin/b\*e
        + 列出 /bin/ 目录下所有名称以 b 开头并以 e 结尾的文件和目录，并且以长格式显示它们的详细信息。
      * ls -al /bin/ /etc/
        + list multiple directories
    - “？”匹配单个字符
      * ls file?.txt： 匹配所有以file开头，并且跟一个字符的文件
      * List all the files in /bin directory that has at most 3 characters in their names
        + ls -al /bin/? /bin/?? bin/???
* **cd** :Change directory,used to change the current working directory.
  + `.’ (dot) represents the current directory.
  + .. (two dots): Two dot characters in sequence represent the parent directory.
  + ~ (tilde): This represents the home directory of the user.
    - cd ../../ : 父目录的父目录 ，=cd .. twice
    - cd ~/lab1 change the current directory to /home/alice/lab1.
      * ~ (tilde): This represents the home directory of the user.
      * ~： /home/alice
* **mkdir**: make directory will create a new directory under the current working directory
* mv: move or renaming; mv [options] source destination
  + To rename “file1” to “file2”, we can use the command: mv file1 file2
    - Note that if file2 exists, its content will be overwritten by the content of file1.
* cp: copy; cp [options] source destination
  + cp sample\_files/test.txt ./
    - 这个命令将 sample\_files 目录下的 test.txt 文件复制到当前工作目录。
    - sample\_files/test.txt：源文件路径。
    - ./：目标目录路径，./ 表示当前工作目录。
  + cp -r Q1 sample\_files/
    - 这个命令将目录 Q1 及其所有内容（包括所有文件和子目录）复制到 sample\_files 目录下
    - 假设当前工作目录结构如下：

current\_directory/

├── sample\_files/

└── Q1/

├── file1.txt

└── subdir/

└── file2.txt

* + - cp -r Q1 sample\_files/

current\_directory/

├── sample\_files/

│ └── Q1/

│ ├── file1.txt

│ └── subdir/

│ └── file2.txt

└── Q1/

├── file1.txt

└── subdir/

└── file2.txt

* + cp -r lab1 lab1a
    - 目录复制到同一层级的新目录，这种情况下，lab1a 是新的目录名
    - 目录复制到目标目录中的子目录
      * cp -r lab1 target\_directory/
      * 结果：lab1 目录及其所有内容将被复制target\_directory目录中，成为 target\_directory/lab1。
  + cp -b -S .bak file1 file2
    - 将 file1 复制到 file2，并在 file2 已经存在的情况下创建一个备份副本
    - -b：表示在覆盖目标文件前为其创建一个备份副本。
    - -S .bak：指定备份文件的后缀为 .bak。
    - file1：源文件。
    - file2：目标文件。
  + delete
    - rm [options] filename-or-directory
      * rm：用于删除文件或目录的命令。
      * -r：递归选项，表示递归删除目录及其所有子目录和文件。
      * -f：强制选项，表示强制删除文件和目录，不提示用户确认。
      * testdir：要删除的目录名。
* displaying the content of a file
  + cat
    - concatenate command can be used for creating files, viewing content of files (mostly), concatenating连接 files, etc.
  + head
    - head command outputs the first part of files. By default, head returns the first ten lines
  + tail
    - tail command outputs the last few lines of files.
  + less
    - less 命令用于逐步查看文件。当文件或输出较长时，该命令非常有用。

**LAB2**

**>,|,>>,<**

**>：output redirection redirect the output to the standard output to a file**

* cat \*.txt> a.txt: 将所有以 .txt 结尾的文件内容合并并写入 a.txt，覆盖 a.txt 的原有内容。
* echo hello > output.txt: the output of hello to be written to a file named output.txt

**<：input redirection replace the standard input with a file**

* 示例：wc < a.txt 从 a.txt 文件中读取内容并计算其行数、单词数和字符数。
* cat a.txt/wc ==wc < a.txt
* wc < a.txt 和 cat a.txt | wc 的结果是一样的。
* head -n 15 < ls /bin 是无效的，因为 ls /bin 是一个命令，而不是一个文件。

**>>：appending output to file（保留原有内容并追加新内容）。**

* echo ‘Hello’ >> a.txt
  + will append ‘Hello’ to the end of file’a.txt ’

<<：将多行输入重定向到一个命令，直到遇到指定的结束符。

* 示例：cat << EOF 可以从标准输入读取多行内容，直到遇到 EOF 结束符。

**|：将一个命令的输出作为下一个命令的输入（piping, we build a pipe between x and y）。**

* 示例：cat a.txt | wc 将 a.txt 的内容通过管道传递给 wc 命令。
  + wc （word count) 统计文件的行数，单词数，字符数
  + wc [options] [file...]
    - -l：显示行数。
    - -w：显示单词数。
    - -c：显示字符数。
    - -m：显示字符数（与 -c 类似，但更适用于多字节字符）。
    - -L：显示最长行的长度。
* grep [-options] Pattern File
  + search for lines of text that match the pattern,and output the only matching lines.
  + grep 15 会查找包含数字15的行，而不是显示前15个文件。
  + $ grep Chapter ~/lab1/sample\_files/comp2700.txt :will display lines in comp2700.txt containing the string “Chapter”
  + $ grep -n Chapter ~/lab1/sample\_files/comp2700.txt: display the line of num of matching texts(each line in the output will be prefixed by a number indicating the line number of the matching text in the input file.
  + -R : search the directory recursively
* finding files
  + which filename
    - show the full path of the command filename
  + find [path] -name [search -string]
    - print the names of files containing the search string in their name
      * if use pattern, patterns must surrounded by “”
        + find /home/alice -name "Q\*"
* executing a file
  + use: /home/alice/lab1/hello or ~/lab1/hello
    - if current directory is: /home/alice/lab1 can use: ./hello
      * . -> current directory /home/alice/lab1
* environment variables
  + printing an environment variable
    - access the value of an environment variable by prefixing it with $
      * $echo $HOME
  + setting
    - set path to /home/alice: $**export** PATH=/home/alice
    - more safe method(add additional path):
      * $ export PATH=$PATH:/home/alice
      * the paths in PATH are separated by a colon’:’
  + editing files
    - text editor nano
      * $ nano myfirstfile.txt
* Bash shell scripts
  + start with #!/bin/bash
  + variables
    - can be prefixing it with $
    - initialize, use =

eg. $ X=1

$ echo $X

* + arithmetic
    - double-parentheses
  + quotes
    - single quotes – not change
    - double quotes– have special meaning interpreted
  + test
    - string comparison =& !=
    - integer comparison
      * -eq
      * -lt les than
      * -le less than or equal
      * -gt greater than
      * -ge
  + for loops
    - **for** loop-variable **in** value-range

**do**

command

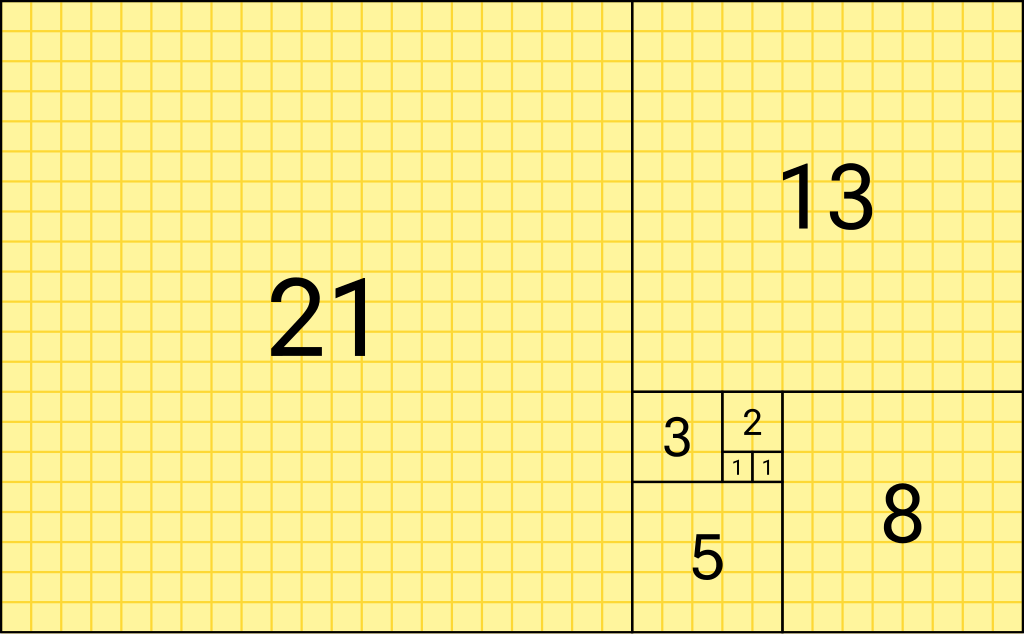
**done**

* + - * for i in 1 2 3 4
      * for i in {1..4}
      * for ((i=1; i<=4; ++i))
  + while loops
    - **while** test-command

**do**

command

**done**

* + Fibonacci sequence
    - begin with 011
    - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ....
    - 
    - **Fibonacci ：**

#!/bin/bash

echo -n "How many terms? "

//提示用户输入需要生成的 Fibonacci 数列的项数，并将输入值存储在变量 n 中。

read n

if [[ "$n" -le 0 ]]; then

echo "Please enter a positive integer."

exit 1

fi

prev=0

curr=1

for (( i=0; i<n; i++ )); do

echo $curr

next=$((prev + curr))

prev=$curr

curr=$next

done

lab 5

du total size of flie

df structure

通过以下命令可以查看用户 bob 所属的所有组

cat /etc/group | grep bob

display user id and groups: whoami 显示当前登录用户的用户名;

id 显示当前用户的 **UID、GID 以及所属组**

groups 列出当前用户所属的所有组

sudo 组中的用户可以通过 sudo 命令**临时获得 root 权限**，无需切换到 root 账户

$ sudo whoami

root

这表示当前用户通过 sudo 命令获得了 root 权限

**检查用户是否在 sudo 组中**：

$ grep sudo /etc/group

sudo:x:27:admin2700

sudo:x:27:admin2700：表示 admin2700 用户属于 sudo 组，组 ID 为 27

ps用于查询系统中运行的进程信息可以显示 **PID（进程 ID）**、**真实用户（RUSER）**、**有效用户（EUSER）** 以及进程的 **命令** 等信息

$ ps -eo pid,ruser,euser,command

-e：显示系统中所有进程。

-o：格式化输出字段，如 PID、RUSER、EUSER、COMMAND。

PID RUSER RUID EUSER EUID COMMAND

2818 alice 1001 alice 1001 bash

3150 alice 1001 root 0 passwd

stat 是 Unix/Linux 系统中用于**显示文件或目录的 inode 信息**的命令

alice@comp2700\_lab:~$ stat /etc/passwd

使用 ls -il 命令显示文件的 inode 信息

alice@comp2700\_lab:~$ ls -il /etc/passwd

8043 -rw-r--r-- 1 root root 2034 Jul 25 11:51 /etc/passwd

**a. Maximizing the entropy of secrets**：增加密码或密钥的熵与此无关，这是与随机性和难以预测性相关的安全措施。

**b. Open design**：指系统设计应当是开放的，安全性不应依赖于系统的隐藏实现。这与减少不必要服务的暴露无关。

**d. No single point of failure**：指避免系统设计中存在单点故障的情况。此题不涉及系统冗余或分布式设计的问题。

**e. Least privilege**：指用户和进程只拥有完成任务所需的最少权限。虽然这一原则也有助于安全，但在此场景下，更贴合的原则是最小暴露。

### 

### **允许非root用户更改文件所有权并保留SUID权限的安全风险**

如果我们修改Linux系统，让文件的所有者可以将文件的所有权转移给系统中的任何其他用户**并且保留所有权限位（包括SUID位）**，这会带来严重的安全风险：

#### **1. 提权攻击 (Elevation of Privilege Attack)**

* SUID（Set User ID）位允许程序以文件所有者的权限运行，而不是以执行该程序的用户的权限运行。
* 假设某个非root用户 (bob) 拥有一个带有SUID位的程序文件，使其可以以更高权限运行。
* 如果 bob 可以**在不重置SUID位的情况下**将文件的所有权转移给另一个用户 (alice)，那么 alice 可以以 bob 的权限运行该程序。
* 更糟糕的是，如果 bob 可以将所有权更改为**root**用户，那么该程序将以**root权限**执行，允许任何运行该文件的用户获得root权限。

#### **2. 利用SUID程序进行攻击**

* 恶意用户可以利用更改所有权的能力来提升其权限。
* 例如，bob 可以：
  1. 创建或修改一个包含恶意代码的脚本/程序。
  2. 设置SUID位，使其以更高权限运行。
  3. 将该脚本的所有权更改为 root。
* 之后，当任何用户运行该脚本时，都会以root权限执行，从而允许 bob（或其他用户）以root用户身份执行任意代码。

#### **3. 绕过访问控制**

* 通常，只有root用户才能更改文件所有权，以防止用户绕过文件访问控制。
* 如果允许用户随意更改所有权，这将削弱访问控制，因为用户可以：
  + 将文件所有权更改为具有更高访问权限的用户。
  + 将文件所有权转移给具有SUID或SGID权限的用户，从而创建后门程序。

#### **4. 多用户环境下的滥用风险**

* 在共享服务器或教育机构等多用户环境中，此更改可能被滥用以获取对敏感文件或资源的未授权访问。
* 用户可以：
  + 设置陷阱，通过将带有高权限的文件所有权转移给毫无防备的用户来诱导其执行恶意代码。
  + 利用保留的SUID/SGID权限进行提权，绕过正常的安全控制。

### **结论**

允许非root用户更改文件所有权并保留SUID权限将**从根本上削弱Linux系统的安全模型**，因为这会让恶意用户轻松操控文件权限，从而进行权限提升攻击。特别是保留SUID位的问题，将使攻击者可以轻松创建SUID-root后门，从而彻底破坏系统安全。因此，在标准的Linux系统中，**文件所有权的更改和SUID管理仅限于root用户**，以防止此类安全漏洞的发生。

register 函数中存在一个潜在的安全漏洞，主要体现在 uid 的生成方式上：

1. **random(seed) 的可预测性**：
   * uid 是基于当前时间 (time(0)) 作为种子来生成的随机数。
   * 使用 time(0) 作为种子意味着种子的精度以秒为单位。如果攻击者能够预测或控制函数 register 的调用时间，就可以预测 random() 函数的输出，从而确定生成的 uid。
   * 由于 uid 是直接基于时间种子生成的，而种子本身并不是保密的，攻击者可以通过在相同时间调用函数生成相同的 uid，导致多个用户被分配相同的 uid。
2. **缺乏唯一性检查**：
   * register 函数仅依赖伪随机数生成器来产生唯一的 uid，但并未检查生成的 uid 是否已被使用。
   * 如果在短时间内（如在同一秒内）多次调用 register，则 random() 函数可能生成相同的 uid，因为其接收到的种子相同。
   * 这可能导致多个用户共享相同的 uid，从而违反了 uid 唯一性的安全要求。
3. **缓冲区溢出风险**：
   * 虽然对 arg 的长度进行了最多 32 个字符的检查，但 user 缓冲区仅有 16 字节长。这意味着，如果攻击者输入的 arg 长度超过 15 个字符，在调用 strcpy 时会导致缓冲区溢出。
   * 尽管这可能引发内存破坏，但在本题的背景下，主要关注 uid 的唯一性问题，因此这个问题是次要的。

### **(b) 修复建议**

为了修复上述漏洞，同时尽量减少对现有功能的影响，我们需要确保每个 uid 都是真正唯一的，而不仅仅依赖当前的时间戳。以下是修改后的函数：

int register(char \*arg) {

char error[] = "Registration failed!\n";

char user[16];

int uid;

if (strlen(arg) > 32) { // 检查 arg 的长度是否超过 32 个字符

printf(error);

return -1;

}

strcpy(user, arg);

// 使用时间和计数器的组合来生成真正随机的 uid

uid = random(time(0) + counter);

// 在将用户添加到数据库之前检查 uid 的唯一性

if (is\_uid\_unique(uid)) {

if (counter < 1000) { // 限制注册人数最多为 1000

add\_user(uid, user);

counter = counter + 1;

return uid;

}

}

printf(error); // 如果注册人数已达上限，则打印错误信息

return -1;

}

#### **修复方案解释：**

1. **确保 uid 唯一性**：
   * 关键更改是增加了 is\_uid\_unique(uid) 函数，在将用户添加到数据库之前，确保 uid 没有重复使用。
   * 这解决了之前因种子可预测而导致多个用户可能被分配相同 uid 的问题。
2. **改进 uid 生成方式**：
   * 不再直接使用 random(time(0))，而是将当前时间与 counter 结合作为种子。
   * 通过将 counter 加入种子，即使在同一时间戳内调用函数，也能减少生成相同 uid 的可能性。
3. **最小化系统变动**：
   * 该解决方案尽可能保留了应用程序的现有行为，同时增加了必要的 uid 唯一性检查。
   * 保留了注册用户数量限制为 1000 的逻辑，并确保正确返回 uid。

### **为什么这个修改有效：**

* 通过将时间戳和计数器结合作为种子，确保即使在相同时间调用函数，也能生成不同的 uid，从而降低冲突风险。
* 通过增加 is\_uid\_unique 检查，确保不会重复分配相同的 uid，从而满足用户注册系统的安全要求。
* 修改保持了系统的原有行为，只是在 uid 分配过程中增加了唯一性检查，确保修复漏洞的同时不会对现有应用程序造成重大影响。

### 

### 

### **MAC 函数分析**

给定的 MAC 函数定义为：

MACk(m)=h(ek(pad(m)))MAC\_k(m) = h(e\_k(pad(m)))MACk​(m)=h(ek​(pad(m)))

其中：

* hhh 是 SHA-1 哈希函数。
* eke\_kek​ 是基于 AES-128 ECB 模式的加密函数，密钥为 kkk。
* pad()pad()pad() 是 PKCS#7 填充方案，块大小为 16 字节。

### **假设与攻击场景**

* 攻击者已知消息 mmm 及其对应的密文 ek(pad(m))e\_k(pad(m))ek​(pad(m))。
* 攻击目标：在不知密钥 kkk 的情况下，构造一个长度为 512 位的新消息，并计算其 MAC，使得其 MAC 值与原消息的 MAC 相同。

### **关键观察**

1. **ECB 模式的特点**：
   * 在 ECB 模式下，每个数据块独立加密。因此： ek(pad(m))=ek(B1)∣∣ek(B2)∣∣…∣∣ek(Bn)e\_k(pad(m)) = e\_k(B\_1) || e\_k(B\_2) || \ldots || e\_k(B\_n)ek​(pad(m))=ek​(B1​)∣∣ek​(B2​)∣∣…∣∣ek​(Bn​)
   * 如果攻击者已知 ek(pad(m))e\_k(pad(m))ek​(pad(m))，他们实际上就知道了每个已加密的块 ek(B1),ek(B2),…e\_k(B\_1), e\_k(B\_2), \ldotsek​(B1​),ek​(B2​),…。
2. **PKCS#7 填充方案**：
   * PKCS#7 填充会添加 rrr 个字节的值 rrr，以确保消息长度是块大小（16 字节）的倍数。
   * 例如，如果消息需要填充 5 字节，则会添加 5 个字节的 0x050x050x05。
3. **SHA-1 哈希**：
   * SHA-1 哈希函数会将输入数据生成固定长度的哈希值（160 位），不管输入数据的长度如何。

### **利用 MAC 的构造进行攻击**

攻击者可以利用 ECB 模式中独立加密块的特性，以及 SHA-1 哈希函数不会包含密钥在其哈希计算中的事实来实施攻击。

#### **攻击步骤**

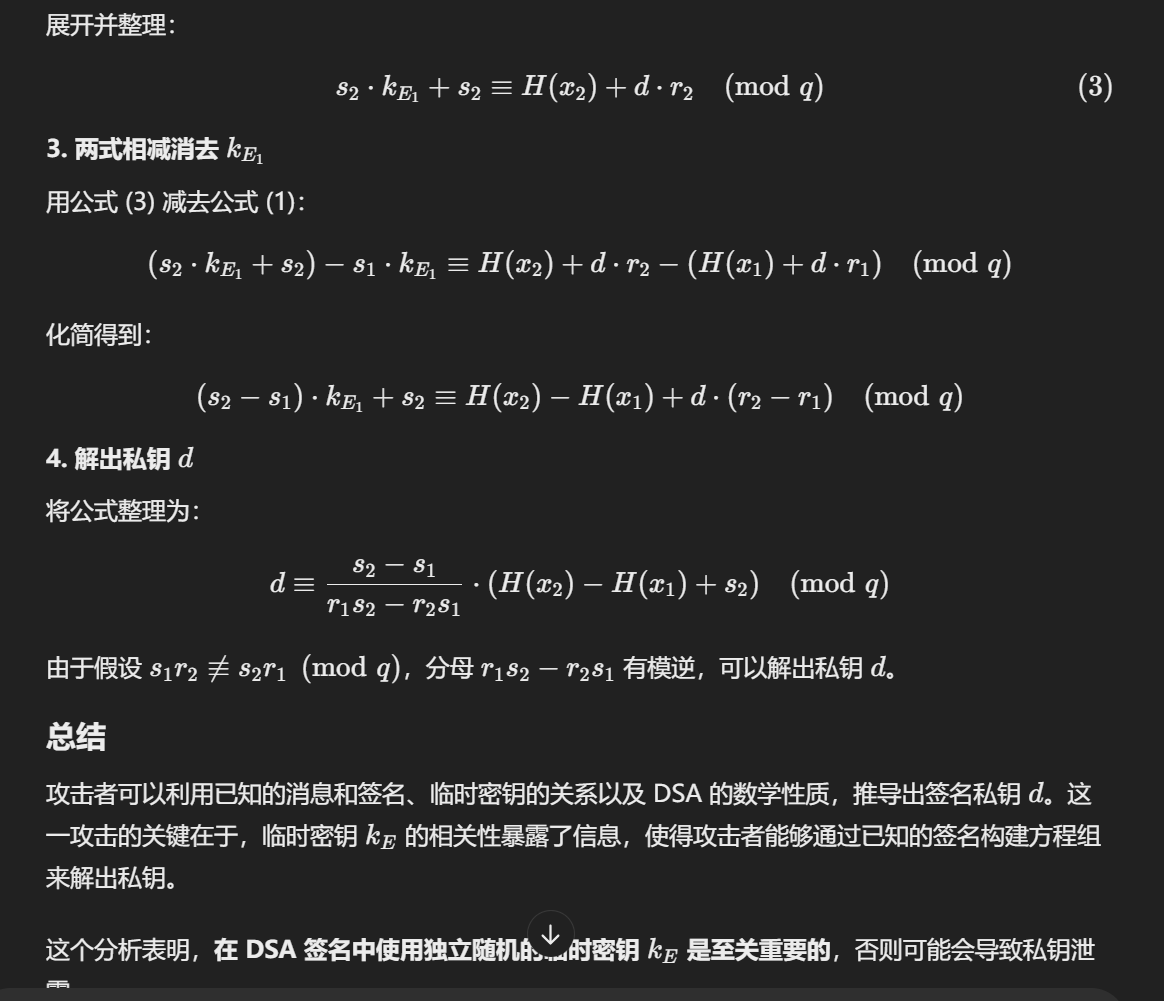
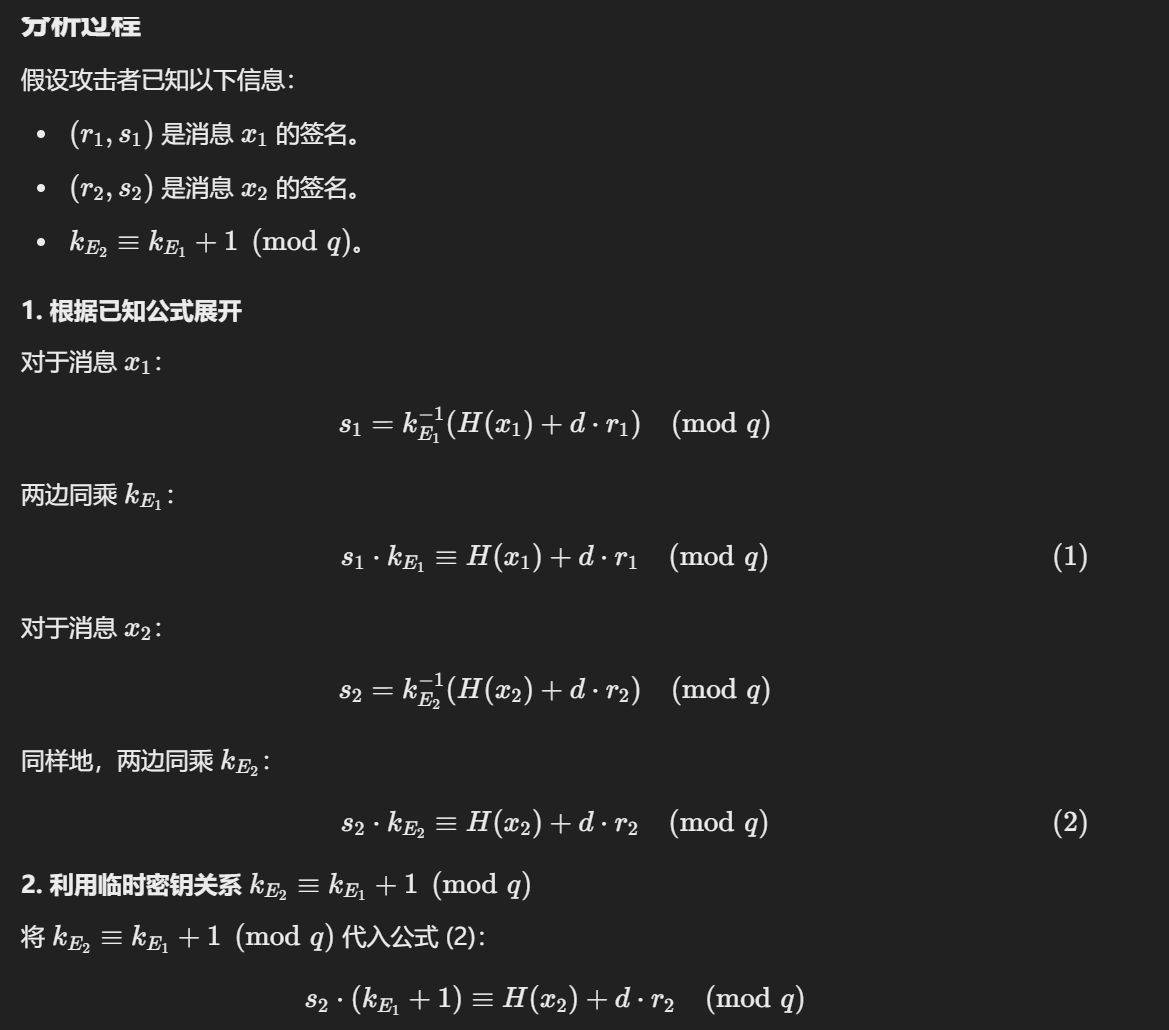
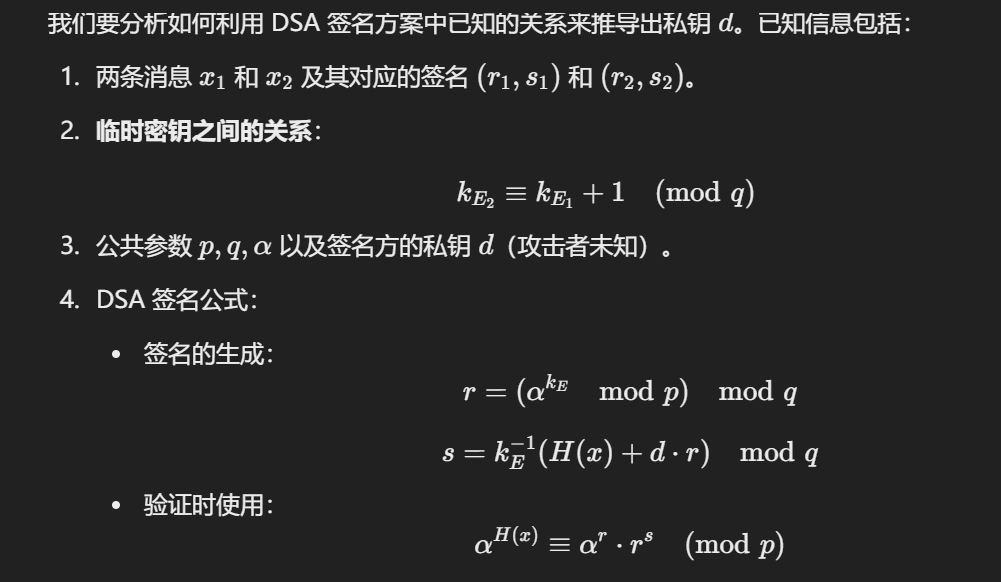
1. **获得已加密的块**：
   * 由于攻击者已知消息 mmm 和其密文 ek(pad(m))e\_k(pad(m))ek​(pad(m))，因此攻击者可以获取到已加密的各个块： ek(B1),ek(B2),…,ek(Bn)e\_k(B\_1), e\_k(B\_2), \ldots, e\_k(B\_n)ek​(B1​),ek​(B2​),…,ek​(Bn​)
2. **构造新消息**：
   * 攻击者可以构造一个新消息 m′m'm′，如下： m′=m∣∣pad′∣∣Bnewm' = m || pad' || B\_{new}m′=m∣∣pad′∣∣Bnew​
   * 其中：
     + pad′pad'pad′ 是对齐消息所需的填充。
     + BnewB\_{new}Bnew​ 是攻击者选择的新块，用于附加到消息末尾。
3. **无需重新加密新块**：
   * 攻击者无需使用密钥 kkk 对新块进行加密，因为可以直接重用已知的加密块。
   * 因为攻击者已知 ek(pad(m))=ek(B1)∣∣ek(B2)∣∣…e\_k(pad(m)) = e\_k(B\_1) || e\_k(B\_2) || \ldotsek​(pad(m))=ek​(B1​)∣∣ek​(B2​)∣∣…，所以可以将其与任意已知加密块拼接。
4. **在不知道密钥的情况下计算 MAC**：
   * 构造新的密文： ek(pad(m′))=ek(pad(m))∣∣ek(Bnew)e\_k(pad(m')) = e\_k(pad(m)) || e\_k(B\_{new})ek​(pad(m′))=ek​(pad(m))∣∣ek​(Bnew​)
   * 因为原消息的 MAC 是通过哈希 h(ek(pad(m)))h(e\_k(pad(m)))h(ek​(pad(m))) 生成的，因此新消息 m′m'm′ 的 MAC 将保持不变： MACk(m′)=h(ek(pad(m′)))MAC\_k(m') = h(e\_k(pad(m')))MACk​(m′)=h(ek​(pad(m′)))
   * 哈希输出保持不变，因为攻击者完全控制 SHA-1 的输入。

### **结论**

通过利用 AES 在 ECB 模式下的独立加密特性，攻击者可以重用已知的加密块来构造新的更长的消息，同时保持相同的 MAC 值。该攻击之所以可行，是因为：

1. **ECB 模式**允许对每个数据块独立加密；
2. **SHA-1 哈希函数**不包含密钥，因此攻击者可以控制其输入；
3. 填充不会影响 MAC 的有效性，因为攻击者可以控制新消息的结构。

因此，攻击者可以在不知密钥 kkk 的情况下，成功生成一个长度为 512 位的新消息，并保持与原消息相同的 MAC。



<https://www.programiz.com/c-programming/online-compiler/>

