# SKEY 协议设计实验

**[实验目的]**

1.掌握身份认证协议的原理和基本思想。

2.掌握 SKEY 协议的机制和实现方法。

**[实验预备]**

1.SKEY 协议的作用。

2.SKEY 协议的安全性分析。

3.SKEY 协议的实现过程。

**[实验内容]**

1. 分析 SKEY 协议的实现过程。
2. 用程序设计语言将算法过程编程实现。
3. 演示 SKEY 协议的身份鉴别过程。

**[实验步骤]**

1. 预习 SKEY 协议的机制。
2. 选择和实现相应的摘要算法 MD5 或 SHA。
3. 写出算法流程，用程序设计语言将协议过程编程实现。
4. 验证 SKEY 协议的身份鉴别过程。

**[问题讨论]**

1. 分析 SKEY 的安全性；
2. 分析实验中在编辑、编译、运行等各环节中所出现的问题及解决方法。

# [SKEY 协议简介]

在计算机接入因特网之后,面临各种攻击的威胁,其中一种威胁可能是搭线窃听者通过偷听你在网上传输的登陆口令或 ID,并在适当的时候 利用这些信息,对你的计算机进行登陆并阅读或修改你的重要信息.因为传统的身份认证技术采 用口令认证机制来验证用户的合法性,其用户名和口令都是以明文的方式在网上传输,一旦攻击者截获用户名和口令,系统就被攻破了.

为了保证口令的私密性,使用了一种容易加密,却很难解密的被称作“单向散列”的方法来处理口令.换言之, 操作系统本身并不真的知道您的口令,它只知道口令经过加密的形式,这使 得口令不再以明文的方式进行传输,要想获取口 令对应“明文” 的唯一办法就只有采用暴力方法在口令可能的区间内进行穷举.

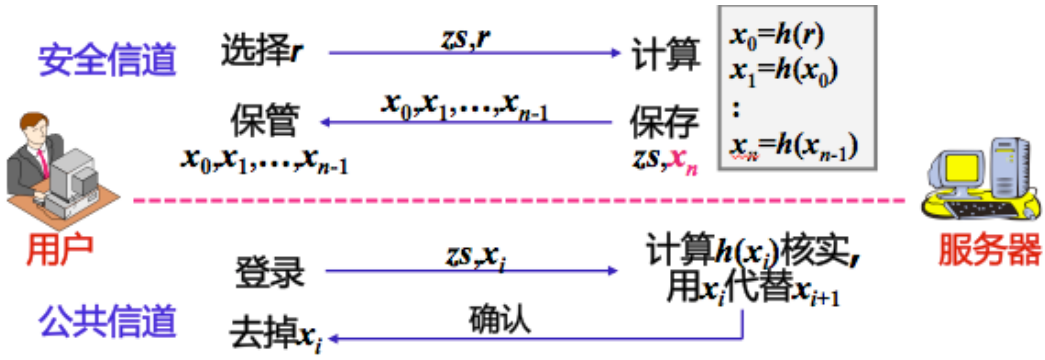
一次性口令认证技术是一种比传统口令鉴别技术更加安全的身份认证技术,其基本思想是在登录过程中加入不确定因素,使每次登录时计算的密码都不相同,系统也 做同样的运算来验证登录,以提高登录过程安全。SKEY 协议一次性口令系统是基于单向 Hash 功能的一次性密码管理方式。

# [SKEY 协议分析]

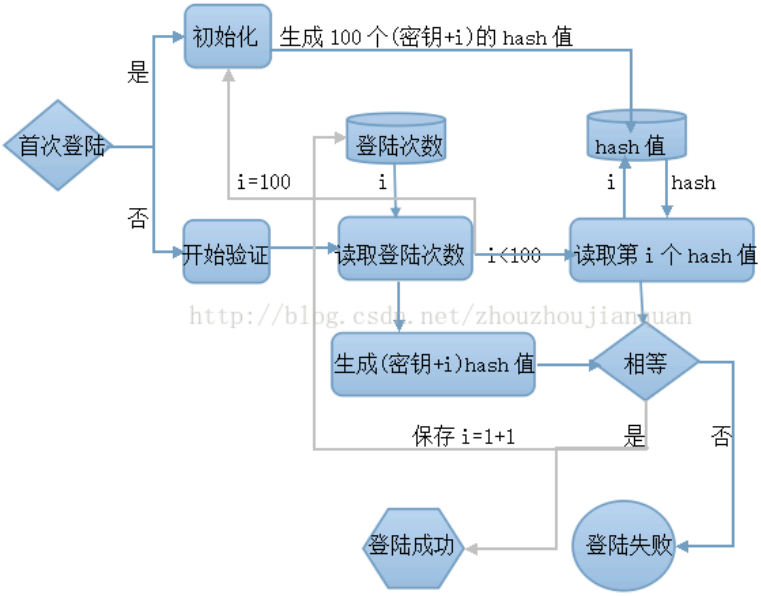
SKEY 身份认证解决方案，可以有效解决重放攻击。重放攻击是指攻击者通过某种方式在网络连接中获取他人的登陆账户与口令，然后利用它多某个网络资源的访问权限。而现在 SKEY 协议分配给访问者的口令每次都不同，所以，就可以有效解决口令泄漏问题。因此，可以避免重放攻击。

SKEY 是一次性口令的一个工具.它是一个基于客户端\服务器的应用程序.首先在服务器端可以用为每个用户建立一个 SKEY 客户,这个命令需要指定一个秘密口令,然后就可以为客户端的用户产生一次性口令列表.当用户与服务器进行连接时就可以按照一次性口令列表中的口令顺序输入自己的密码,下次再连接时候密码就换成了列表中的下一个。

原理图如下：



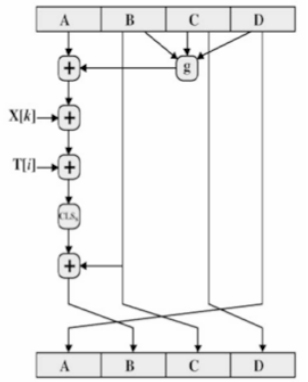

算法设计流程图如下：






## [MD5 算法]

本实验中 hash 函数采用 MD5 算法，实现流程图如下：




Function g g(b,c,d)
1 F(b,c,d) (b∧c)∨(b̄∧d)
2 G(b,c,d) (b∧d)∨(c∧d̄)
3 H(b,c,d) b⊕c⊕d
4 I(b,c,d) c⊕(b∨d̄)

轮数X[k]的k值
1 0,1,2,3,…,15
2 1,6,11,0,…,12
3 5,8,11,14,…,2
4 0,7,14,5,…,9

MD5（单向散列算法）的全称是 Message-Digest Algorithm 5（信息-摘要算法），经 MD2、MD3 和 MD4 发展而来。MD5 算法的使用不需要支付任何版权费用。

## 1.MD5 功能：

输入任意长度的信息，经过处理，输出为 128 位的信息（数字指纹）；
不同的输入得到的不同的结果（唯一性）；
根据 128 位的输出结果不可能反推出输入的信息（不可逆）；

## 1.MD5 用途：

a、防止被篡改：

1）比如发送一个电子文档，发送前，我先得到 MD5 的输出结果 a。然后在对方收到电子文档后，对方也得到一个 MD5 的输出结果 b。如果 a 与 b 一样就代表中途未被篡改。2）比如我提供文件下载，为了防止不法分子在安装程序中添加木马，我可以在网站上公布由安装文件得到的 MD5 输出结果。3）SVN 在检测文件是否在 CheckOut 后被修改过，也是用到了 MD5.

b、防止直接看到明文：

现在很多网站在数据库存储用户的密码的时候都是存储用户密码的 MD5 值。这样就算不法分子得到数据库的用户密码的 MD5 值，也无法知道用户的密码(其实这样是不安全的，后面我会提到)。（比如在 UNIX 系统中用户的密码就是以 MD5（或其它类似的算法）经加密后存储在文件系统中。当用户登录的时候，系统把用户输入的密码计算成 MD5 值，然后再去和保存在文件系统中的 MD5 值进行比较，进而确定输入的密码是否正确。通过这样的步骤，系统在并不知道用户密码的明码的情况下就可以确定用户登录系统的合法性。这不但可以避免用户的密码被具有系统管理员权限的用户知道，而且还在一定程度上增加了密码被破解的难度。）

c、防止抵赖（数字签名）：

这需要一个第三方认证机构。例如 A 写了一个文件，认证机构对此文件用 MD5 算法产生摘要信息并做好记录。若以后 A 说这文件不是他写的，权威机构只需对此文件重新产生摘要信息，然后跟记录在册的摘要信息进行比对，相同的话，就证明是 A 写的了。这就是所谓的“数字签名”。

## 3.MD5 算法过程：

对 MD5 算法简要的叙述可以为：MD5 以 512 位分组来处理输入的信息，且每一分组又被划分为 16 个 32 位子分组，经过了一系列的处理后，算法的输出由四个 32 位分组组成，将这四个 32 位分组级联后将生成一个 128 位散列值。

第一步、填充：如果输入信息的长度(bit)对 512 求余的结果不等于 448，就需要填充使得对 512 求余的结果等于 448。填充的方法是填充一个 1 和 n 个 0。填充完后，信息的长度就为 N*512+448(bit)；

第二步、记录信息长度：用 64 位来存储填充前信息长度。这 64 位加在第一步结果的后面，这样信息长度就变为 N*512+448+64=(N+1)*512 位。

第三步、装入标准的幻数（四个整数）：标准的幻数（物理顺序）是（$A=(01234567)_{16}$，$B=(89ABCDEF)_{16}$，$C=(FEDCBA98)_{16}$，$D=(76543210)_{16}$）。如果在程序中定义应该是（A=0X67452301L，B=0XEFCDAB89L，C=0X98BADCFEL，D=0X10325476L）。有点晕哈，其实想一想就明白了。

第四步、四轮循环运算：循环的次数是分组的个数（N+1）

本实验中 MD5 算法实现：

```
public class MD5 {
  public static  String myMD5(String src) {

    byte[] input = src.getBytes();

    try {
      MessageDigest digest = MessageDigest.getInstance("MD5");
      //就是找到要加密的字符串,放置到update()中等待digest()方法进行产出
      //就像是在机器的两头:一头的将原材料放进去，一头是产品的输出
      //获取输入
      digest.update(input);
      //获得产出
      input = digest.digest();

      //下面就是进行十六进制的转换
      int length = input.length;
      StringBuffer strBuff = new StringBuffer();
      for(int i = 0;i<length;i++)
      {
        //将字符转变成对应的ASSIC值
        int val = ((int)input[i])&0xff;
        //转变成对应的值后小于4位
        if(val<16)
        {
          strBuff.append("0");
        }
        strBuff.append(Integer.toHexString(val));
      }

      return strBuff.toString();
    } catch (NoSuchAlgorithmException e) {
      e.printStackTrace();
    }

    return src;
  }
}
```

# ［SKEY 协议设计流程］

1.输入随机数 R，计算机计算 hash(R), hash(hash(R)), hash(hash(hash(R)))等等 100 次。调用 x1 ，x2 ，x3 …..x100 来记录这些数。计算机打印出这些数的列表，Client 把这些数放入口袋妥善保管，计算机也顺利地在登录数据库中 Client 的名字后面存储 x100 的值。

```
public class skey {

    String[] hashLink = new String[10];
    String[] keys = new String[9];
    String saved = "";
    int flag = 0;// 游标
    File file = new File("Key.txt");//哈希link保存

    public void Init() throws Exception{

        String init = JOptionPane.showInputDialog(" 初始化： ");

        String string = init;
        for (int i = 0; i < 10; i++) {

            String tem = MD5.myMD5(string);
            string = tem;
            hashLink[i] = string;
            System.out.println(string);
        }

        saved = string;// 保留最后一个

        PrintStream ps = new PrintStream(new FileOutputStream(file));

        for (int i = 0; i < 9; i++) {
            keys[i] = hashLink[8 - i];// 保管密钥，反向hash链
            ps.println(keys[i]);
        }
```

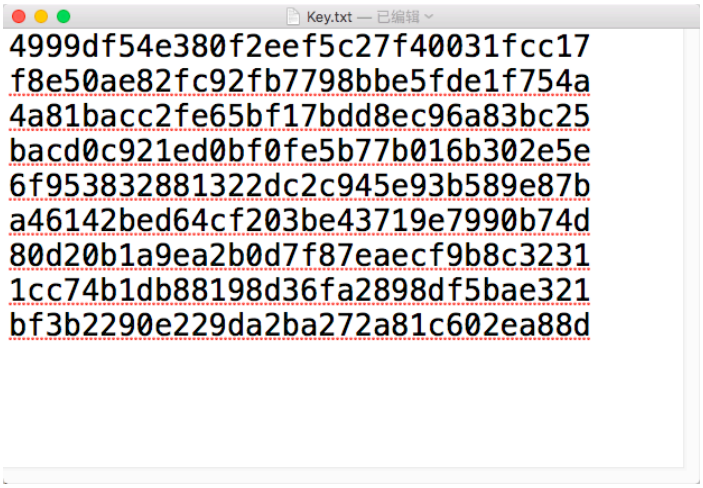
2.设计客户端登陆界面，当 Client 第一次登录时，输入名字和 x99，计算机计算 hash（x99），并把它和 x100 比较，如果它们匹配，那么证明身份是真的。然后，计算机用 x10 代替数据库中的 x99。

```
public void Login() throws Exception{

    while (true) {
        String key = JOptionPane.showInputDialog(null, "确认登陆吗?");
        if (saved.equals(MD5.myMD5(key)))
            {flag++;
            JOptionPane.showMessageDialog(null, "第" + flag + "次登陆成功!");}
        else
            {
            JOptionPane.showMessageDialog(null, "登陆失败!");
            continue;
            }

        if (flag == 9) {
            JOptionPane.showMessageDialog(null, "动态口令已使用完毕");
            break;
        }

        saved = key;//服务器更换口令

    }

}
```

# [程序测试分析]

1.输入随机数 R，初始化


2.本地保存的密钥（为演示方便只生成十个密钥，除去保存在服务器的一个还剩下九个，也就是能登录九次）



3.第一次登陆时输入第一条口令，实际上保存在文件中的口令是反序状态。登录成功后服务器替换原口令



4.登录成功后如果再使用第一次的口令或输入错误口令会导致登录失败



5.当所有口令使用完后，给以提示并不允许继续登陆

# [问题讨论]

1. 分析 SKEY 的安全性；

SKEY 是比较安全的，因为每个数只被用一次，并且这个函数是单向的，所以攻击者不可能得到任何有用的信息。同样的，数据库对攻击者也毫无用处。当然，当用户用完了的列表上面的数后，必须重新初始化系统。总之 SKEY 协议可以有效解决重放攻击，并且 SKEY 协议分配给访问者的口令每次都不同，所以可以有效解决口令泄漏问题。

2. 分析实验中在编辑、编译、运行等各环节中所出现的问题及解决方法。

开始使用 hash 链在登陆时没有设置在文件中反序保存，导致登陆失败，后来发现应该从后往前输入，最终将文件中保存的口令反序以便登陆。