如何进行双向认证

1. 浏览器发送一个连接请求给安全服务器。
2. 服务器将自己的证书，以及同证书相关的信息发送给客户浏览器。
3. 客户浏览器检查服务器送过来的证书是否是由自己信赖的CA中心（如沃通CA）所签发的。如果是，就继续执行协议;如果不是，客户浏览器就给客户一个警告消息：警告客户这个证书不是可以信赖的，询问客户是否需要继续。
4. 接着客户浏览器比较证书里的消息，例如域名和公钥，与服务器刚刚发送的相关消息是否一致，如果是一致的，客户浏览器认可这个服务器的合法身份。
5. 服务器要求客户发送客户自己的证书。收到后，服务器验证客户的证书，如果没有通过验证，拒绝连接;如果通过验证，服务器获得用户的公钥。
6. 客户浏览器告诉服务器自己所能够支持的通讯对称密码方案。
7. 服务器从客户发送过来的密码方案中，选择一种加密程度最高的密码方案，用客户的公钥加过密后通知浏览器。
8. 浏览器针对这个密码方案，选择一个通话密钥，接着用服务器的公钥加过密后发送给服务器。
9. 服务器接收到浏览器送过来的消息，用自己的私钥解密，获得通话密钥。
10. 服务器、浏览器接下来的通讯都是用对称密码方案，对称密钥是加过密的。

如何防止字典攻击和穷举攻击

- 增大攻击的时间需求

1. 通过增大密钥位数或加大解密（加密）算法的复杂性来对抗穷举攻击
2. 增大两次解密尝试的间隔时间, 或在多次尝试后锁定账户

- 动态口令

书写动态口令认证技术内容

- 存储口令hash值
- 输入id
- i为口令计数值
- client计算i-1次hash迭代
- server计算i次hash迭代
- 匹配

```
var i int
func Client(hashkey int) ans int {
    ans = hashkey
    k := 0
    for k < i - 1 {
        k++
        ans = hash(ans)
    }
}
func Server(key int) ans int {
    ans = key
    k := 0
```

```
    for k < i {
        k++
        ans = hash(ans)
    }
}
func hash() int {
    hash函数
}

func judge(a, b int) bool {
    if a == b {
        return true
    } else {
        return false
    }
}

judge(Server(key), Client(hashkey))
key由client发送
hashkey由server发送
```