在使用，因为其他较强的权能字仍然需要该校验值。

新的权能字被被发送回请求进程。现在用户可以在消息中附加该权能字发送到朋友处。如果朋友打开了应该被关闭的权限位，服务器就会在使用权限字时检测到，因为f的值与错误的权限位不能对应。既然朋友不知道真正的校验字段，他就不能伪造与错误的权限位相对应的权能字。这种方法最早是由Amoeba系统开发的，后被广泛使用（Tanenbaum等人，1990）。

除了特定的与对象相关的权限（如读和执行操作）外，权能字中（包括在核心态和密码保护模式下）通常包含一些可用于所有对象的普通权限。这些普通权限有：

1) 复制权能字：为同一个对象创建新的权能字。

2) 复制对象：用新的权能字创建对象的副本。

3) 移除权能字：从权能字列表中删去登录项；不影响对象。

4) 销毁对象：永久性地移除对象和权能字。

最后值得说明的是，在核心管理的权能子系统中，撤回对对象的访问是十分困难的。系统很难为任意对象找到它所有显著的权能字并撤回，因为它们存储在磁盘各处的权能字列表中。一种办法是把每个权能字指向间接的对象，而不是对象本身。再把间接对象指向真正的对象，这样系统就能打断连接关系使权能字无效。（当指向间接对象的权能字后来出现在系统中时，用户将发现间接对象指向的是一个空的对象。）

在Amoeba系统结构中，撤回权能字是十分容易的。要做的仅仅是改变存放在对象里的校验字段。只要改变一次就可以使所有的失效。但是没有一种机制可以有选择性地撤回权能字，如，仅撤回John的许可权，但不撤回任何其他人的。这一缺陷也被认为是权限系统的一个主要问题。

另一个主要问题是确保合法权能字的拥有者不会给他最好的朋友1000个副本。采用核心管理权能字的模式，如Hydra系统，这个问题得到解决。但在如Amoeba这样的分布式系统中却无法解决这个问题。

另一方面，权能字非常漂亮地解决了移动代码的沙盒问题。当外来程序开始运行时，给出的权能字列表里只包含了机器所有者想要给的权能，如在屏幕上进行写操作以及在刚创建的临时目录里读写文件的权利。如果移动代码被放进了自己的只拥有这些有限权能的进程中，就无法访问其他任何资源，相当于被有效地限制在了沙盒里。这种方法不需要修改代码，也不需要解释性执行。当运行的代码拥有所需的最少访问权时，符合了最小特权规则，这也是建立安全操作系统的方针。

总之，ACL和权能字具有一些彼此互补的特性。权能字相对来说效率较高，因为进程在要求“打开由权能字3所指向的文件”时无须任何检查。而采用ACL时需要一些查验操作（时间可能很长）。如果系统不支持用户组的话，赋予每个用户读文件的权限就需要在ACL中列举所有的用户。权能字还可以十分容易地封装进程，而ACL却不能。另一方面，ACL支持有选择地撤回权限，而权能字不行。最后，如果对象被删除时权能字未被删除，或者权能字被删除时对象未被删除，问题就会发生。而ACL不会产生这样的问题。

9.3.4 可信系统

人们总是可以从各种渠道中获得关于病毒、蠕虫以及其他相关的消息。天真的人可能会问下面两个问题：

1) 建立一个安全的操作系统有可能吗？

2) 如果可能，为什么不去做呢？

第一个问题的答案原则上是肯定的。如何建立安全系统的答案人们数十年前就知道了。例如，在20世纪60年代设计的MULTICS就把安全作为主要目标之一而且做得非常好。

为什么不建立一个安全系统是一个更为复杂的问题，主要原因有两个。首先，现代系统虽然不安全但是用户不愿抛弃它们。假设Microsoft宣布除了Windows外还有一个新的SecureOS产品，并保证不会受到病毒感染但不能运行Windows应用程序，那么很少会有用户和公司把Windows像个烫手山芋一样扔掉转而立即购买新的系统。事实上Microsoft的确有一款SecureOS（Fandrich等人，2006），但是并没有投入商业市场。

第二个原因更敏感。现在已知的建立安全系统仅有的办法是保持系统的简单性。特性是安全的大敌。系统设计师相信（无论是正确还是错误的）用户所想要的是更多的特性。更多的特性意味着更多的复杂性，更多的代码以及更多的安全性错误。