摘 要

随着 Internet 在全世界的普及，在加强人们沟通的同时，也暴露出了严重的安全隐患，于是网络安全也就自然越来越引起人们的关注。而现在广泛使用的 IPv4 协议，最初并没有考虑到安全性，随着安全问题的日益突出，新一代网络安全标准 IPSec（IP Security）应运而生。

IPSec 是 IETF 为保证 IP 层机密性、完整性、可认证性而制定的一个开放的安全标准框架，它是一组网络安全协议的集合。文中对 IPSec 协议体系进行了阐述。

路由器作为 IP 网络的核心设备，在网络互联中的地位举足轻重。要想保护网络安全，就必须把好路由器这一关。因此，路由器的安全措施势在必行。经过几年的研究与实践，2002 年 5 月，实验室正式启动国家“863”重点项目－“高性能 IPv6 路由器协议栈软件”。IPSec 作为网络层的安全协议，成为路由器协议栈软件中的一分子。在自主开发的路由器协议栈软件上实现 IPSec，正是本论文构建的基础。

我们在论文中给出了路由器的软件功能结构，提出将 IPSec 作为一个协议功能软件来实现的设计思想,并选择实时嵌入式操作系统作为路由器实施的软件平台；推出了基于 IPSec 协议的安全路由器构架，针对 VxWorks 和实时 Linux 这两种操作系统，分别给出在 IPv4 和 IPv6 双协议栈下 IPSec 的具体实施方案,并分模块重点介绍了 VxWorks 下 IPSec 实现的细节问题。最后是对所实现系统的调试与测试结果，针对路由器中 IPSec 实现的不足之处，给出改进建议并提出发展方向。

关键词：IPSec　路由器　实时嵌入式操作系统

Abstract

With the prevalence of Internet over the World, it strengthens the communication of people but also exposes serious secure hidden trouble. So the problem of network security attracts more and more attention of people. But IPv4 which is popularly applied now is a protocol with no consideration of security. With increasing prominence of security problem, IPSec-the new security standard for Internet, was born with time.

IPSec is a open standard security frame constituted by IETF for IP layer's confidentiality、integrality and authentication. It is a suit of network security protocols. We expatiate on IPSec protocol architecture in this paper.

As the core device of IP network, Router's status holds the balance in network's interlinkage. In order to ensure network security, Router must be well safeguarded. So, security measures should be implemented on Router. After a lot of research and practice, the "863 Program" – "the Protocol Suite Software of High Performance of IPv6 Router" was started up in our laboratory, May 2002. As network security protocol, IPSec becomes member of the Router's Protocol Suite. Realization of IPSec on the Router's Protocol Suite which is developed by ourselves is the base of this paper.

we give out low end Router's architecture of its software function, and bring forward the design thought that realize IPSec as a protocol's functional software, what's more, select real-time embedded operation system as the platform of Router's software.What's more, we put forward secure Router's architecture based on IPSec, give out the concretely implementing scheme aiming at VxWorks and real-time Linux operation systems, emphasize the detailed problems of IPSec realization under VxWorks.The last part is the debugging and testing results of the system.Finally, we point out the deficiency of IPSec realization and come to some improved measures and developing direction in the future.

Keywords: IPSec(IP Secutity) Router RT-embedded OS

独 创 性 说 明

本人声明所呈交的论文是我个人在导师的指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京交通大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签 名：宇瑞敏 日 期：2004.3.5

关于论文使用授权的说明

本人完全了解北京交通大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。论文中所有创新和成果归北京交通大学 IP 网络实验室所有。未经许可，任何单位和个人不得拷贝。版权所有，违者必究。

签 名：宇瑞敏 导师签名：梁雷 日 期：2004.3.5

未经作者、导师同意
勿全文公布

第一章　绪 论

1.1　选题背景及意义

1.1.1　Internet 安全

随着计算机网络的不断发展，全球信息化已成为人类发展的大趋势。但由于计算机网络具有联结形式多样性、终端分布不均匀性和网络的开放性、互连性等特征，致使网络易受黑客、怪客、恶意软件和其他不轨的攻击。在 Internet 所受到的攻击中，最严重的包括 IP 哄骗（入侵者伪造假的 IP 地址，并对基于 IP 认证的应用程序进行哄骗）和各种的窃听和嗅探网包（攻击者在信息传输过程中非法截取如登录口令或数据库内容一类的敏感信息）。所以网上信息的安全和保密是一个至关重要的问题。对于军用的自动化指挥网络、C3I 系统和银行等传输敏感数据的计算机网络系统而言，其网上信息的安全和保密尤为重要。

因此，上述的网络必须有足够强的安全措施，否则该网络将是个无用、甚至会危及国家安全的网络。但是无论是在局域网还是在广域网中，都存在着自然和人为等诸多因素的脆弱性和潜在威胁。故此，网络的安全措施应是能全方位地针对各种不同的威胁和脆弱性，这样才能确保网络信息的保密性、完整性和可用性。

1.1.2　IPSec 提出

我们可以在 Internet 上使用许多机制来提供安全，具体使用何种机制依赖许多决定因素，这些决定因素大致上可以分为三类[10]：威胁原型，即什么人会使用什么机制来攻击什么资源；保护粒度，即具体保护的对象；实现层，即安全机制的实现层次。一般来说，如果安全机制能和协

议结合严谨，就是安全的、清晰的、有效的设计方案，并且安全机制实现的层次越低，能够保护的协议就越多，对实现该安全机制的系统的修改也就越小，而随着 Internet 的发展，IP 协议的重要性越来越突出，IP 协议将是未来各种网络交换的中心，大部分的网络技术都将是基于 IP 协议的。而且，通常一个系统的安全并不仅仅是基于单独一个单元，而是一些单元综合作用的结果。在用来阻止较低层次的攻击来说，IP 层次确实是一个正确的层次，这些攻击如上所述，由于简单易行，它们在所有的网络攻击中占有非常大的比例。所以，在 IP 层实现的安全机制才能更好的满足当前的安全需要，而现有的互联网协议标准（IPv4）未提供任何的安全特性，IP 数据包本质上是不安全的，很容易被拦截、篡改、伪造或重播。为此，IETF 专门制订了新一代的网络层安全标准－IPSec（IP Security）。

早在 1994 年，互联网体系机构理事会（Internet Architecture Board，IAB）曾发表了一篇关于《互联网体系结构中的安全问题（RFC1636）》的报告。报告中陈述了人们对安全的渴望，并阐述了安全机制的关键技术，其中包括保护网络构架免受非法监视及控制，以及保证终端用户之间使用认证和加密技术进行安全交易等。由此，Internet 工程任务组 IETF 于 1994 年开始了一项 IP 安全工程，专门成立了 IP 安全协议工作组 IPSEC，来制定和推动一套称为 IPSec 的 IP 安全协议标准，其目标就是把安全集成到 IP 层，以便对 Internet 的安全业务提供低层支持。从 1995 年开始，IETF 着手研究制定一套用于保护 IP 通信的 IP 安全协议，并于 1995 年 8 月公布了一系列关于 IPSec 的建议标准。

如今 IPSec 协议已成为新一代 Internet 的安全标准，它可以“无缝”地为 IP 引入安全特性，并对数据源提供身份验证，数据安全完整性检查以及机密性保证机制，可以防范数据收到来路不明地攻击。由于是处于

网络层地安全协议，IPSec 协议可被上层地任何协议所使用，如 TCP、UDP、ICMP、BGP 等等。IPSec 的设计既适用于 IPv4 又适用于 IPv6，它在 IPv4 中作为一个建议的可选服务，对 IPv6 是一项必需支持的功能。

作为目前唯一能为任何形式的 Internet 通信提供安全保障的协议，IPSec 的问世立即受到了全世界的瞩目，并被业界普遍接受和应用。各种应用程序可以享用 IP 层提供的安全服务和密钥管理，而不必设计和实现自己的安全机制，因此减少密钥协商的开销，也降低了产生安全漏洞的可能性。IPSec 可连续或递归应用，在路由器、防火墙、主机和通信链路上配置，实现端到端安全、虚拟专用网络（VPN）和安全隧道技术。而且，IPSec 技术的出现为 VPN 的实现提供了一种优越的途径，它已经成为构建虚拟专用网的基础。

1.1.3　TCP/IP 各层安全机制比较

由于 TCP/IP 协议本身不具备安全功能，人们就根据实际应用需求对 TCP/IP 协议各层逐步进行安全增强，开发了多种安全协议和机制：

1、防火墙（Firewall）

防火墙是建立在内部网和外部网之间的安全监控系统，是系统的第一道防线，用于实现访问控制，即阻止外部入侵者进入内部网，而允许内部网用户访问外部网络，从而防止互联网的损坏（如病毒或资源盗用）波及内部网络。防火墙按照系统管理员预先定义好的规则来控制数据包的进出，它具有以下特点：

1）内部网与外部网交流的所有信息都必须通过它；

2）只有经过授权许可的通信业务才被允许通过；

3）防火墙自身对外部具有一定的抗入侵能力。

设计防火墙的目的有四点：

1）它限制访问者进入一个被严格控制的点；

2）它防止进攻者接近防御设备；

3）它限制访问者离开一个被严格控制的点；

4）检查、筛选、过滤和屏蔽信息流中的有害服务，防止对计算机系统进行蓄意破坏。

当前实现“防火墙”功能的主要技术有：数据包过滤、应用网关和代理服务器等。Internet 的“防火墙”技术目前已经比较成熟，简单的防火墙技术可在路由器上直接实现，而专用的防火墙提供更加可靠的网络安全控制方法。

2、IP 包过滤（IP Packet Filtering）

包过滤技术是在网络层对数据包进行分析、选择，选择的依据是系统内设置的过滤逻辑，称为访问控制列表。通过检查数据流中的每一个数据包的源地址、目的地址、所用端口号、协议状态等因素，或它们的组合来确定是否允许数据包通过。

3、网络地址转换（NAT）

网络地址转换可以动态的改变 IP 报文中的源 IP 地址和（或）目的 IP 地址，以达到地址重用或地址隐藏的目的。

4、IP 安全体系结构（IPSec）

IPSec 协议是为 IPv4 和 IPv6 提供可互操作的、高质量的基于密码的安全性。提供的安全服务系列包括接入控制、无连接完整性、数据源验证、抗重播保护、机密性（加密）、有限通信流机密性。这些服务在 IP 层提供，保护 IP 和/或上层协议。这些目标通过使用两种通信安全协议（即 AH 和 ESP）和密钥管理程序及协议来实现。

5、SOCKS

是一个需要认证的防火墙协议。SOCKS 协议的优势在于访问控制，

因此适合用于安全性较高的 VPN。SOCKS v5 在 OSI 模型的会话层控制数据流，它定义了非常详细的访问控制，在客户机和主机之间建立一条虚电路，可由此对用户的认证进行监视和访问控制。用 SOCKSv5 的代理服务器可隐藏网络地址结构。如果 SOCKv5 同防火墙结合起来使用，数据包经一个唯一的防火墙端口到代理服务器，代理服务器然后过滤发往目的计算机的数据，这样可以防止防火墙上存在的漏洞。因为 SOCKSv5 通过代理服务器来增加一层安全性，因此其性能往往比低层协议差。

6、安全套接字层（SSL）

SSL 是在 Internet 基础上提供的一种保证私密性的安全协议。它能使客户/服务器应用之间的通信不被攻击者窃听，并且始终对服务器进行认证，还可选择对客户进行认证。SSL 协议要求建立在可靠的传输层协议之上，SSL 协议的优势在于它是与应用层协议独立无关的。高层的应用层协议能透明的建立于 SSL 协议之上。SSL 协议在应用层协议通信之前就已经完成加密算法、通信密钥的协商以及服务器认证工作。在此之后应用层协议所传送的数据都会被加密，从而保证通信的私密性。

7、应用层网关

应用层网关是在网络的应用层上实现协议过滤和转发功能。它针对特定的网络应用服务协议使用指定的数据过滤逻辑规则，并在过滤的同时，对数据包进行必要的分析、记录和统计，形成报告。

8、AAA 认证技术

AAA 即 Authentication（认证）、Authoriztion（授权）、Accounting（记帐）的缩写 。AAA 认证技术是设计用来动态配置远程用户的每条线路或每项服务的认证、授权、记帐模型，通过创建访问控制列表来定义认证、授权和记帐的类型，然后把这些方法列表应用到具体服务或接口上。

AAA 认证技术应用于内部网络，在内部网络中设置拨号服务器或网络访问服务器向远程用户提供拨号访问服务，通过在拨号服务器或网络访问服务器及内部网的安全服务器上运行的 AAA 认证软件对远程拨号用户进行身份认证、授权及记帐，抵御通过电话线绕过防火墙对内部网络的攻击。

各种安全机制的安全性比较如表 1-1 所示：

功能 / 措施	访问控制	加密	认证	完整性检查	PFS	地址隐藏	会话监控
IP 包过滤	Y	N	N	N	N	N	N
NAT	Y	N	N	N	N	Y	Y（连接）
IPSec	Y	Y（包）	Y（包）	Y（包）	Y	Y	N
SOCKS	Y	N	Y（客户）	N	N	N	Y（连接）
SSL	Y	Y（数据）	Y（系统）	Y	N	N	Y
应用网关	Y	通常没有	Y（用户）	Y	通常没有	Y	Y（连接/数据）
AAA 服务器	Y(用户)	N	Y（用户）	N	N	N	N

表 1-1　TCP/IP 各种安全机制的比较[10]

从上面地比较中，可以发现 IPSec 具备地安全功能最多，是一个比较好地网络安全解决方案。

目前，组建大型信息网络的关键技术是 TCP/IP 网络互联和路由技

术，特别是在 Internet 中，路由器起着重要的作用。而在网络安全日益重要的今天，“安全路由器”成为保证互联网（同时也包括 Intranet 和 Extraner）信息安全的关键设备之一。由于信息安全的特殊性和重要性，出于对国外安全产品可能留有“陷门”的考虑，国家要求关键部门使用的安全路由产品必须有自主版权。因此，自主开发集常规路由和安全加密功能于一体的“安全路由器”，有极其重要的意义。我们选择 IPSec 作为路由器上实现安全措施的协议，即在完成普通路由器功能的基础上实施 IPSec 协议簇，它可以：防止虚假路由信息的接收；防止路由器的非法接入；对路由信息和 IP 数据包进行加密保护；对复杂网络加密的正确性和系统的可用性进行检查。

1.2 国内外 IPSec 研究现状

从 1993 年初 IETF 成立了 IPSec 工作组开始进行研究与探讨 IPSec 到现在已近 11 年，在这 11 年中间，IPSec 由最初的雏形发展到现在形成了较为完善的体系方案：1998 年 11 月 IPSec 工作组公布了 RFC2401-RFC2412,不仅将 ISAKMP 和 Oakley 制定成为标准，并且修改了 RFC1825-RFC1829 做为新的 RFC 公布，以 RFC2401 为基础的这一系列文档成为 IPSec 协议较为完整的文档。并且从 98 年末开始，对 IPSec 开始逐步完善，这些完善工作包括：定义、修改和完善 SA、IKE 协议及所用到的加密和鉴定算法；新成立 IPSP(IP Security Policy)工作组来解决 IPSec 策略管理机制及其协议的修改、完善和标准化工作等等。并且还在继续完善过程中。

作为网络层的安全标准，IPSec 一经提出，就引起计算机网络界的注意，几乎世界上所有的计算机公司都宣布支持这个标准，并且不断推出

自己的应用产品。目前，国外个别公司已做成了产品，但基本都是针对IPv4的，且都与操作系统紧密捆绑：一类是操作系统型，作为主机操作系统的一部分被实现。目前很多大家所熟知的操作系统，如自由操作系统（包括 Linux、BSD 和 Unix）、商业操作系统（如 Microsoft 的windows2000、Windriver的Vxworks及Sun的Solaris8）中都集成了IPSec；另一类是路由器型，作为专用路由器或路由器软件的安全功能被实现，主要的路由器厂家（如 Cisco、Intel、Nortel Networks）生产的专用 VPN网关都支持 IPSec。此外，国外还有采用硬件实现 IPSec 的技术，如NetOctave 公司推出的 NSP3000B-IPsec Security Accelerator Board，声称提供全软件支持，包括API等，并支持BSD/OS 4.2; Red Hat Linux 6.2, 7.1, and 7.2, and VxWorks 5.4 操作系统。NetScreen Technologies 的 NetScreen系列的高性能的硬件防火墙产品，速度可达吉比特/秒，适合于企业级或服务提供商的宽带数据应用。关于不同的 IPSec 实现间的互操作性，目前还没有令人满意的测试报告。但国外基于 IPv6 协议的 IPSec 仍处于开发研究阶段，尚未产品化。

国内已经开展了对 IPSec 的研究，不少公司虽对 IPSec 进行了实现，但也只是针对 IPv4，且并不完善。而且，网关的性能降低严重，同类产品性能降低在 30%左右。这将导致极大的安全问题。整体而言，国内的IPSec 的整体水平仍处于研究实验阶段，尤其尚无 IPv6 的成熟技术和产品出现。

但是由于标准提出的时间很短，而且其中又有一个重要组成部分没有标准化，因此尽管产品种类很多，但真正合格的产品却很少。目前世界上最权威的 IPSec 产品评测中心为 ICSA（国际计算机安全协会）实验室，至 2001 年 5 月 25 日，只有 35 个产品通过了它的测评。这 35 个产品都是国外的产品，没有我国的产品。

对国外通过 ICSA 测评的产品和国内通过中国国家信息安全测评认证中心计算机测评中心和公安部计算机信息安全产品质量监督检验中心测评的产品的分析、比较，国外的产品在安全性、可扩展性、使用简单和性能价格比、协议实现的完整性、系统的配置、系统的管理、日志及报表、系统自身的安全性、性能、CA 认证等许多方面要成熟和优于国内产品。目前我国产品主要存在以下问题：操作系统的安全问题，这些产品所基于的操作系统都是国外的产品，操作系统的安全性控制在他人手里；大多是与防火墙整合在一起，重在防火墙功能，而对 IPSec 的实现相对较为简单和不完善，因此几乎无法作为真正的 VPN 网关使用；安全策略系统实现并不完善。

1.3　论文的项目来源及研究目标

2002 年 5 月北京交通大学电子学院 IP 网络实验室签订并启动了国家高技术研究发展计划（863 计划）高性能 IPv6 路由器协议栈软件项目(2001AA121014)。本项目的主要目标是面向下一代互联网的发展趋势与需求，研究高性能路由器 IPv6 协议实现技术，自主开发完成 IPv6 协议栈软件，形成具有自主知识产权的协议软件实现技术，突破高性能路由器研制过程中软件严重滞后的瓶颈，为高性能 IPv6 路由器的研究开发及产业化提供关键技术支撑。其中路由器协议栈软件主要涉及到以下五个协议的实现：路由协议——RIPng，OSPFv3，BGP4+；网管协议——SNMPv3；安全协议——IPSec。本论文构建在此项目的 IPSec 协议实现上，具体来说就是在 VxWorks 和实时 Linux 两种操作系统下实现支持 IPv4 和 IPv6 双协议栈的 IPSec 协议。

1.4 主要工作

在论文中，作者对IPSec协议本身进行了详细的阐述，并围绕着IPSec协议软件在操作系统下的具体实现做了大量的工作，具体来说主要包括以下几个方面：

1、针对互联网的安全现状分析了在 IP 层实施安全的必要性；对现行各种安全措施进行比较，指出IPSec协议的优越性；论述了IPSec安全路由器的意义。

2、在消化吸收 IETF 制定的系列相关标准的基础上，深入剖析了IPSec协议体系，为以后IPSec协议具体实现奠定基础。

3、分析了路由器中软件功能模块组成，对IPSec在路由器协议栈软件中的实施进行了总体设计，提出采用嵌入式实时操作系统作为路由器实施的软件平台。

4、学习了Linux和VxWorks操作系统下的编程，深入分析了这两套操作系统内核网络部分代码的实现。

5、针对VxWorks和实时Linux两种嵌入式操作系统，自行设计了支持IPv4和IPv6双协议栈的IPSec具体方案，并实现了整个协议系统，包括IPSec与操作系统内核的有机结合、套接口的添加，AH和ESP协议对数据包的处理、策略系统SPD、SAD。

6、对用C语言实现的源代码进行编译调试；而且为了验证IPSec协议的功能实现，模拟VPN情况搭建测试环境，制定测试内容，并给出测试结果。

7、分析IPSec在路由器中实现的不足，提出了改进方法，指出以后的发展方向。

论文的结构为：

第一章 绪论　综述选题意义、研究背景及主要工作。

第二章 IPSec 协议体系　分析 IPSec 协议体系的总体结构，介绍 IPSec 的功能、模式、实施方案、策略以及各组成子协议。

第三章 嵌入式实时操作系统 VxWorks　概述路由器实施的软件平台 VxWorks 操作系统，介绍了其系统开发调试环境 Tornado。

第四章 路由器软件总体设计　概述路由器工作原理，分析其软件组成。

第五章 IPSec 在路由器中的实现　给出 IPSec 在路由器中实现总体结构；重点介绍了 VxWorks 操作系统下 IPSec 的实现，针对功能框图对各模块实现进行了详细阐述；概述了实时 Linux 下的实现思想。

第六章 系统调试及测试　结合 Tornado 调试工具给出调试方法，并搭建测试环境进行测试。

第七章 总结和展望　对所作工作进行总结，并针对 IPSec 在路由器中实现的不足，提出改进意见，指出发展方向。

第二章　IPSec 协议体系

2.1　IPSec 综述[9]

IPSec 是一系列基于 IP 网络（包括 Intranet、Extranet 和 Internet）的，由 IETF（Internet Engineering Task Force）正式定制的开放性 IP 安全标准框架，是虚拟专网的基础。它可以保证局域网、专用或公用的广域网及 Internet 上信息传输的安全。IPSec 细则首先于 1995 年在互联网标准草案中颁布。

IP 包本身并不继承任何安全特性。很容易便可伪造出 IP 包的地址、修改其内容、回放以前的包以及在传输途中拦截并查看包的内容。针对这些问题，IPSec 被设计成为能够在 Internet 这样无保护的网络中为 IPv4 和 IPv6 提供安全保证。它在网络层上对实现 IPSec 的设备之间传输的 IP 报文进行保护和认证。它定义了一套默认的、强制实施的算法，以确保不同的实施方案相互间可以共通。IPSec 提供的安全性具体体现在以下方面：

1、数据机密性：确保 IP 报文的内容在传输过程中未被读取。当报文在公网上传送时，未授权方不能读取报文的内容。通过发送方在使用网络传输报文前对报文进行加密，就可以确保攻击者即使截获报文也不能破解报文的内容。

2、数据完整性：接收方在一般情况下对收到的报文不能确定其在传送过程中是否遭到破坏。使用 IPSec 对报文进行认证，就可以确保报文在网络中传送时没有被修改。

3、数据源认证：接收方对报文的源 IP 地址进行认证，以确保报文真的

是由声称的发送者发送的。这项服务是基于数据的完整性。

4、抗重播：确保认证报文没有重复。针对攻击者可能通过重发截获的认证报文来干扰正常的通信，从而导致事务多次执行，使用 IPSec 接收方能够通过对报文序列号的检测来拒绝接收过时的报文或拷贝报文。

IPSec 提供的这些服务都是基于 IP 层的，提供了对 IP 及其上层协议的保护。同时这些服务是可选的，由本地安全策略规定使用这些服务的一种或者多种。

IPSec 为保障 IP 数据报的安全，规定了要保护的通信内容、怎样保护它以及通信数据发给何人。IPSec 的实施可根据不同情况选择，具有一定灵活性，它可保障主机之间、网络安全网关（如路由器或防火墙）之间或主机与安全网关之间的数据包的安全。并且这些实施方式可以嵌套实现。

要想对 IP 数据报或上层协议进行保护，方法是使用某种 IPSec 协议："封装安全载荷（ESP）"或者"验证头（AH）"。其中，AH 可证明数据的起源地、保障数据的完整性以及防止相同数据包的不断回放。ESP 则更进一步，除具有 AH 的所有能力之外，还可选择保障数据的机密性，以及为数据流提供有限的机密性保障。

IPSec 提供的安全服务需要用到共享密钥，以执行它所肩负的数据验证以及（或者）机密性保证任务。当然，采用人工增加密钥的方式，未免会在扩展（伸缩）能力上大打折扣。因此，它定义了一种标准的方法，用以动态地验证 IPSec 参与各方的身份、协商安全服务以及生成共享密钥等等。这种密钥管理协议称为 IKE —亦即"Internet 密钥交换（Internet Key Exchange ）"。

2.2　IPSec 结构

IPSec 是一种协议套件，它包括：AH（验证头）、ESP（封装安全载荷）、IKE（Internet 密钥交换）、ISAKMP/Oakley 以及转码。IPSec 各组件之间的交互方式如下图 2-1 所示。

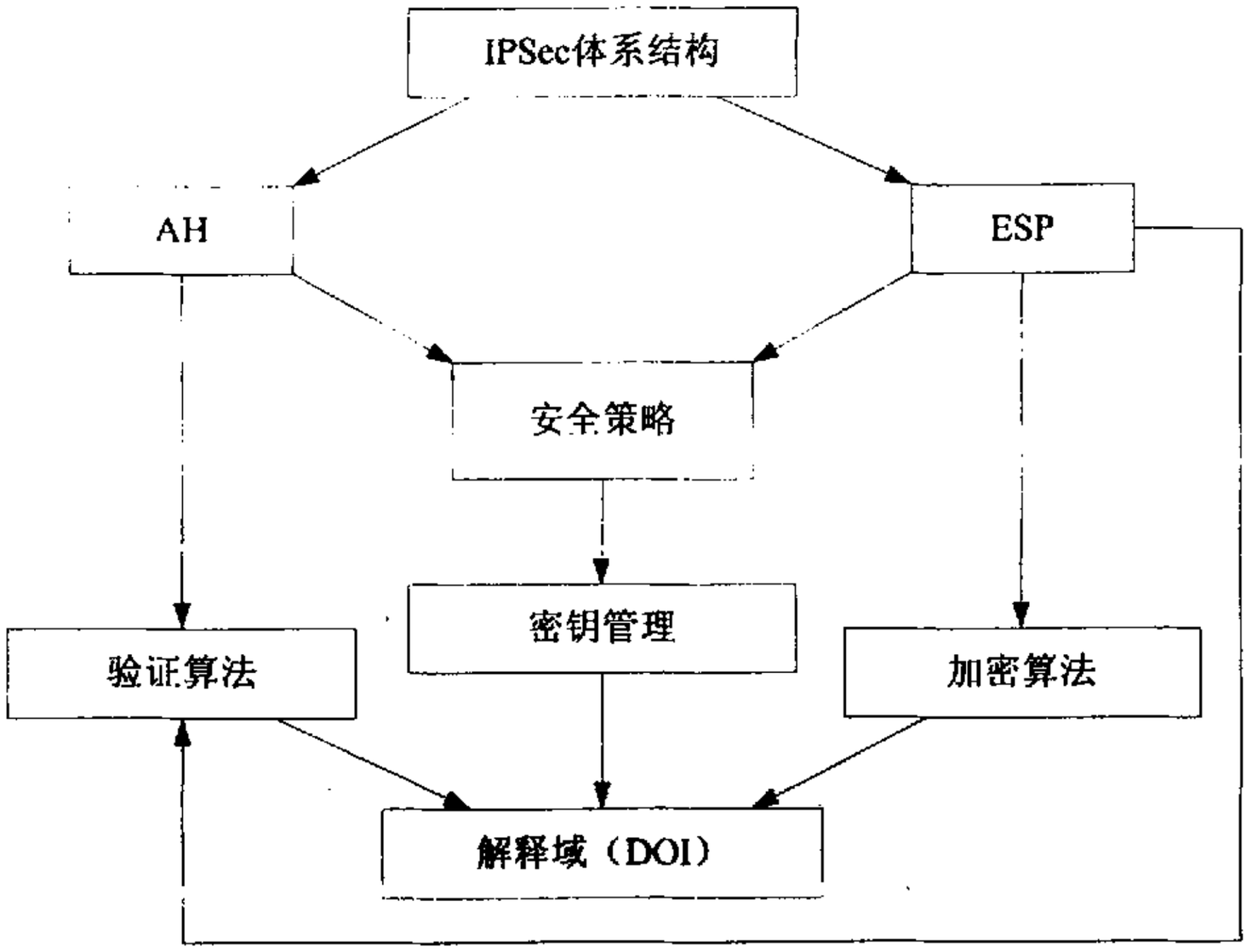


图 2-1　IPSec 体系结构

IPSec 的 AH 和 ESP 协议用于处理报文，与策略数据库进行交互，以便决定为数据流提供何种安全保护，AH 协议提供的服务主要涉及到验证算法，而 ESP 提供的服务涉及到了验证和加密算法。

安全策略的制订和管理是 IPSec 正确实施的前提。策略决定了实体通信的对象、应用于通信数据流的安全服务以及如何对数据报进行处理。此外，策略还规定了密钥管理的方式。

IPSec 的密钥管理有两种方式：人工方式和自动协商方式。鉴于人工

方式的局限性，一般使用 IKE 为 IPSec 协议动态协商生成密钥。其实，IKE 可为任何一种协议商拟密钥，并不仅仅限于 IPSec 的密钥协商。这是通过将 IKE 协商的参数同协议本身分隔开来实现的。协商的参数被归于一个单独的文档内，名为 IPSec 解释域，或者 IPSec DOI。

2.2.1　IPSec 实施[3]

IPSec 的实施灵活性很强，它可在终端主机、网关/路由器或两者中间进行实施和配置。至于它到底在网络的什么地方配置，则由用户对安全保密的要求来决定。

在主机实施有下列好处：保障端到端的安全性、能够实现所有的 IPSec 安全模式、能够逐数据流提供安全保障。主机实施可分成两类：与操作系统（OS）集成实施，将其作为网络层的一部分嵌入；在协议堆栈的网络层与数据链路层之间实施，作为两者之间的一个“楔子”使用，我们称其为“堆栈中的肿块（BITS）”实施方案。

如在路由器中实施，可在网络的一部分中对传输的数据包进行安全保护。在路由器中实施有下列优点：能对通过公用网络在两个子网之间流动的数据提供安全保护；能进行身份验证，并授权用户进入私用网络。路由器实施方案有两种类型：原始实施，它等同于在主机上进行的集成实施方案，在这种情况下，IPSec 是集成在路由器软件中的；线缆中的块（BITW），它等同于 BITS 实施方案，在这种情况下，IPSec 的实现在一个设备中进行，那个设备直接接入路由器的物理接口。

2.2.2　IPSec 模式

IPSec 有两种模式：传送模式和隧道模式。这两种模式的实施方式有所不同：

1、传送模式用来保护上层协议（比如 TCP 和 UDP ），而隧道模式用

来保护整个 IP 数据报。

2、在传送模式中，IP 头与上层协议头之间需插入一个特殊的 IPSec 头；而在隧道模式中，要保护的整个 IP 包都需封装到另一个 IP 数据报里，同时在外部与内部 IP 头之间插入一个 IPSec 头。

3、由构建方法所决定，对传送模式所保护的数据包而言，其通信终点必须是一个加密的终点。在隧道模式，通信终点便是由受保护的内部头指定的地点，而加密终点则是那些由外部 IP 头指定的地点。在 IPSec 处理结束的时候，安全网关会剥离出内部 IP 包，再将那个包转发到它最终的目的地。

4、两种 IPSec 协议（AH 和 ESP）均能同时以传送模式或隧道模式工作。这两种模式的比较如下图 2-2 所示：

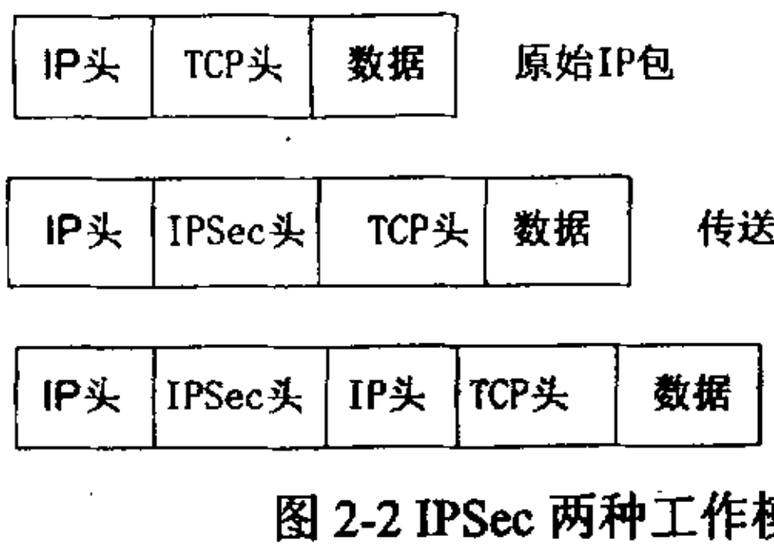


图 2-2 IPSec 两种工作模式比较

2.2.3　安全联盟（SA）

为正确封装及提取 IPSec 数据包，需要采取一套专门的方案，将安全服务/密钥与要保护的通信数据联系到一起；同时要将远程通信实体与要交换密钥的 IPSec 数据传输联系到一起。换言之，要解决如何保护通信数据、保护什么样的通信数据以及由谁来实行保护的问题。这样的构建方案称为“安全联盟（Security Association,SA）”。

SA 是构成 IPSec 的基础。SA 是两个通信实体经过协商建立起来的一种协议，决定了用来保护数据包安全的 IPSec 协议、转码方式、密钥

以及密钥的有效存在时间等。SA 是单向的，为了保证两个主机、两个安全网关之间典型的双向通信安全，需要两个 SA（一个负责一个方向）。而且 SA 还是协议相关的，每个协议都有一个 SA，它为其承载的通信提供的安全服务是通过使用 AH 或者 ESP 赋予的。可以用一个三元组来唯一标识一个 SA，该三元组的元素是：安全参数索引（Security Parameter Index，简称 SPI）、IP 目的地址、安全协议（AH 或 ESP）。

在一个单独的SA上传输的IP数据报只能由一个安全协议提供保护，即 AH 和 ESP 二者只能选其一。有时候一个安全策略要求为一个通信流实施多个服务，这只用一个 SA 是无法实现的。这种情况下，就需要利用多个 SA 来实现所需的安全策略，这就是所谓的"安全联盟束"或"SA 束"，它是指一个 SA 的序列组合。

任何 IPSec 实施方案都会构建一个 SA 数据库（SADB），由它来维护 IPSec 协议用来保障数据包安全的记录。每个 SA 都应在 SADB 中有一条记录项相对应。一条 SA 记录包含：协议、SPI、目的地址、回放窗口大小、序列号、生存期、方向、模式、初始化向量、加密算法与密钥、认证算法与密钥等信息。

在进行 IPSec 处理时用到下列 SADB 字段：

1、序列号计数器：一个 32 位值，被用来在 AH 或 ESP 头中产生序列号字段。（所有实施类型，只用在外出处理）

2、序列号计数器溢出：一个旗标字段，用来检查序列号是否溢出，防止 SA 处理其余的包。（所有实施类型，只用在外出处理）

3、抗回放窗口：用来决定进入的 AH 或 ESP 包是否是回放。（所有实施类型，只用在进入处理）

4、AH 身份验证算法、密钥等。（AH 实施用）

5、ESP 加密算法，密钥。（ESP 实施用）

6、ESP 身份验证算法、密钥等。（如果没用选择身份验证算法，此字段为 NULL）

7、SA 存活时间：它规定了每个 SA 最长能够存在的时间。超过这个时间，那个 SA 便不可继续使用。可采用两种类型的存活时间—软的和硬的。所谓"软存活时间"，是指用它来警告内核，通知它 SA 马上就要到期了。这样一来。在"硬存活时间"到期之前，内核便能及时协商好一个新的 SA。（所有实施类型）

8、IPSec 协议模式：传送模式、隧道模式或通配符。（主机实施必须支持所有模式，网关实施必须支持隧道模式）

9、PMTU 参数：在隧道模式下使用 IPSec 时，必须维持正确的 PMTU 信息，以便对这个数据包进行相应的分段。（只用在外出处理）

2.2.4　安全联盟管理[3]

SA 管理的两大主要任务就是创建与删除。SA 管理既可手工进行，亦可通过一个 Internet 标准密钥管理协议来完成，比如 IKE。为进行 SA 的管理，要求用户应用（含 IKE）的一个接口同内核通信，以便实现对 SADB 数据库的管理。

SA 的创建分两步进行—先协商 SA 参数，再用 SA 更新 SADB。人工密钥协商是必须支持的，在人工密钥协商过程中，通信双方都需要离线同意 SA 的各项参数。SPI 的分配、参数的拟定均是人工进行的。但是非常明显，这个过程容易出错，另一个不足之处是这些 SA 永远不会过期。一旦建立了，它们就会一直呆下去，直到再次以人工方式删除。所以，一般情况下我们都采用动态协商方式创建密钥。IKE 会与目标主机或者途中的主机 / 路由器协商具体的 SA。SA 创建好且加入 SADB 数据库后，保密数据包便会在两个主机间正常地"流动"。

需要删除 SA 可能有多方面的理由：存活时间过期、密钥已遭破解、使用 SA 加密 / 解密或验证的字节数已超过策略设定的某一个阈值、另一端要求删除这个 SA。SA 可以手工删除或通过 IKE 来删除。为降低别人破解系统的可能性，经常需要更新密钥。IPSec 本身没有提供更新密钥的能力。为此，我们必须先删除现有的 SA，再协商并建立一个新 SA。一旦 SA 被删除，它所使用的 SPI 便可重新使用。为避免耽搁通信，必须在现有的 SA 过期之前，协商好一个新的 SA。如果 SA 的存活周期非常短，那么在即将到期的 SA 被删除之前，拥有多个 SA 的通信双方还可利用老的 SA 来保障通信的安全。然而，最好的做法还是重新建立 SA，尽量避免使用以前的 SA。

2.2.5　安全策略

安全策略决定了为一个包提供的安全服务。正如前面提到的那样，对所有 IPSec 实施方案来说，它们都会将策略保存在一个名为 SPD（安全策略数据库）的数据库中。SPD 说明了对 IP 数据报提供何种保护，并以何种方式实施保护，是 SA 处理过程的核心部分。

对一个 IPSec 实施点，无论是正常 IP 包还是 IPSec 包，无论进入包和外出包都需要参考 SPD，从而决定哪些通信流需要进行 IPSec 保护。对于进入或外出的每一份数据报，都可能有三种处理：丢弃、绕过或应用 IPSec。丢弃是指主机或安全网关对数据报不做进一步的处理；绕过是指允许数据报在通过 IPSec 实施点时不进行 IPSec 处理；应用是指数据报在通过 IPSec 实施点时进行 IPSec 处理。对于这些需要处理的数据报，SPD 应明确指出应该提供的安全服务、协议类型、应该使用的算法等等信息。

在实现时，SPD 应包含一个有序的策略项列表。每一个策略项可以

有一个选择符来指定。每一项的内容中包含了对相应的通信流应该实施的策略：绕过、丢弃、处理。如果需要实施处理策略，则从 SPD 项中提

- 取 SA（SA 束）的相关描述：目的地址、SPI、IPSec 协议，从而选择一个 SA 或 SA 束。

一条安全策略可以要求一个或多个 SA（按指定顺序）应用于一个指定的通信流上。因此 SPD 中的策略项必须在必要时保存这些 SA 的顺序要求。在实现时必须可以为输入输出包定义一个 SA 处理序列。对于外出处理，SPD 中的每一项可以由单个的 SA 组成，也可以由一个有序的 SA 束组成。对于进入处理，被应用了多个 SA 的 IPSec 包，将根据三元组：<源数据包的目的地址、协议、SPI>来指定唯一的单个 SA。从而检验进入策略实施的正确性。

同时，SPD 应提供管理接口，便于用户或系统管理员对 SPD 进行维护。该接口允许用户或管理员对每一个进入或外出包都应作出相应的策略说明，并支持通过选择符对 SPD 添加有序的策略项。并且，应用程序可以通过发送请求的方式，为它产生或接受的数据报选择处理策略。系统管理员必须控制用户或应用程序对策略系统的制定。

2.3　IPSec 协议

2.3.1　AH

AH（Authentication Header, 认证头）协议，对IP报文附加加密的校验和来提供报文的认证，实现了数据完整性、数据源认证和抗重放保护服务（抗重放保护服务的选择可以在安全联盟中设置）。它定义在RFC 2402 中。但要注意的是，AH 不对受保护的IP 数据报的任何部分进行加密。AH可用保护一个上层协议（传送模式）或一个完整的IP数据报（隧

道模式）。其间的差别根据受保护的数据报如何应用AH来定。AH协议可以单独实施，也可以与ESP协议一起工作。

2.3.1.1　AH 头格式[2]

典型的AH协议头格式如下图2-3所示，它是由5个固定长度域和一个可变长的认证数据域组成：

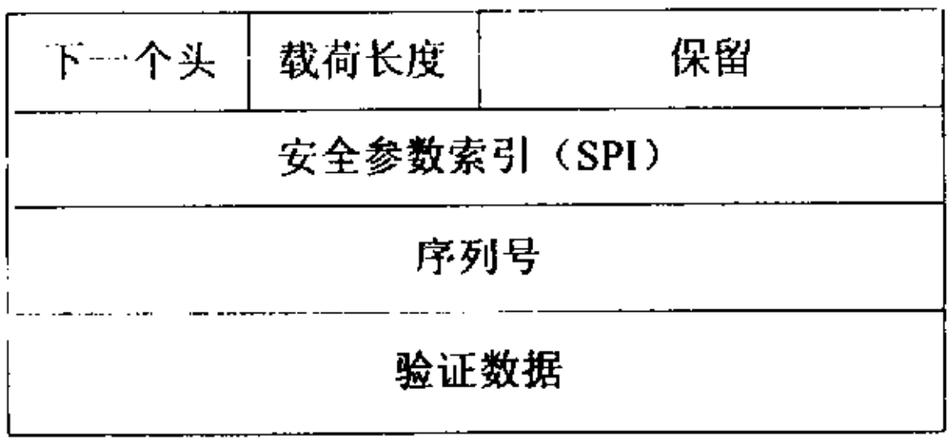


图2-3 AH头格式

各域的意义如下：

1、下一个头：8bit，指出AH后的下一载荷类型。如：AH后是一个ESP载荷，这个域值为50。

2、载荷长度：8bit，表示头本身的长度，以32比特为单位的AH的长度减2。

3、保留：16bit，供将来用，置零。

4、安全参数索引：32bit，其值一般是在IKE交换过程中由目标主机选定的，它与源地址或目的地址及AH或ESP共同唯一标识一个数据报所属数据流的安全联盟（SA）。

5、序列号：32bit，作为单调递增的计数器。SA建立时，发送者和接收者的序列号初始化为0。通信双方每使用一个特定的SA发出一个数据报就将他们相应的序列号加1。序列号用来防止数据报的重播。重播指数据报被攻击者截取并重新传送。AH规范强制发送者总得发送序列号给接收者，而接收者可以选择不使用抗重播功能。如果接收端

主机使用抗重播功能，它使用滑动窗口机制检测重播包。具体的滑动窗口因不同的IPSec实现而不同；然而一般滑动窗口具有以下功能。窗口长度最小为32bit，窗口右边界代表一特定SA所接收到的验证有效的最大序列号，序列号小于窗口左边界的包将被丢弃。将序列号值位于窗口之内的数据包与位于窗口内的接收到的数据包清单比照验证，如果接收到的数据包的序列号位于窗口内且数据包是新的，或者其序列号大于窗口右边界且小于2^{32}，那么接收主机继续处理认证数据的计算。对于一个特定的SA，它的序列号不能循环；所以在一个特定的SA传输的数据包的数目达到2^{32}之前，必须协商一个新的SA以及新的密钥。

6、认证数据：可变长。该认证数据被称为数据报的完整性校验值（ICV）。对于IPv4数据报，这个域的长度必须是32的整数倍；对于IPv6数据报，这个域的长度必须是64的整数倍。用来生成ICV的算法由SA指定。可用的算法因IPSec的实现不同而不同；然而为了保证互操作性，AH强制所有的IPSec实现必须包含两个MAC：HMAC－MD5和HMAC－SHA1。如果一个IPv4数据报的ICV域的长度不是32的整数倍，或一个IPv6数据报的ICV域的长度不是64的整数倍，必须添加填充比特使ICV域的长度达到所需要的长度。

2.3.1.2　AH 算法

AH使用消息认证码（MAC）对IP进行认证。MAC是一种算法，它接收一个任意长度的消息和一个密钥，生成一个固定长度的输出，称作消息摘要。MAC不同于杂凑函数，因为它需要密钥来生成消息摘要，而杂凑函数不需要密钥。最常用的MAC是HMAC。HMAC可以和任何迭代密码杂凑函数（如MD5，SHA1，RIPEMD或Tiger）结合使用，而不用对杂凑函数进行修改。

已经被定义的AH协议认证算法有HMAC-MD5(Hashed Message Authentication Code-Message Digest 5，散列信息认证码-消息摘要5)和HMAC-SHA1（Security Hash Algorithm Version 1，安全散列算法）。通信双方中的一方用密钥对整个IP包进行计算得到摘要值，另一方用同样的密钥和算法进行计算，如果两者的结果相同，则说明数据包在传输的过程中没有被改变，IP包就通过了身份验证。因此，数据的完整性和认证安全得到了保证。

2.3.1.3　AH 模式[2]

AH有两种工作模式：传送模式和隧道模式。不同之处在于它保护的数据要么是一个上层协议，要么就是一个完整的IP数据报。

在传送模式中，AH保护的是端到端的通信。通信的终点必须是IPSec终点。AH 头被插在数据报中，紧跟在IP 头（和任意选项）之后，和需要保护的上层协议之前，对这个数据报进行安全保护。AH在IPv4和IPv6中的位置如图2-4所示：

传输模式下的AH在IPv4中的应用

变长可选域	AH	传输层协议头	传输层协议数据

传输模式下的AH在IPv6中的应用

原始IP头	逐跳、路由、分段、目的等扩展头	AH	传输层协议头	传输层协议数据

图2-4　传送模式下AH在IPv4和IPv6中的位置

其中在IPv6的传送模式下，AH被插在逐跳、路由、分段扩展头的后面；目的扩展头放在AH头的前面或后面均可。

在传送模式下，AH验证整个的IPv4或IPv6头，因而产生了一些局限性。比如采用传送模式AH的主机，不能放在有NAT功能的网关的后面，因为NAT网关会将流出的数据包源地址域的IP地址换为指定的公网地址，重新计算校验和，并将数据包转发给指定的目的地址。这样在对端

的AH完整性校验将失败。

在隧道模式中，它将自己保护的数据报封装起来，AH插在原始的IP头之前，另外生成一个新的IP头放在AH之前。在IPv6中，除了新的IP头外，原来数据报的扩展头也被插在AH前面。“里面的”IP 数据报中包含了通信的原始寻址，而“外面的”IP 数据报则包含了IPSec端点的地址。如图2-5所示：

隧道模式下的AH在IPv4中的应用

新IP头的可选域	AH	原始IP头的可选域	传输层协议头	传输层协议数据

隧道模式下AH在IPv6中的应用

新IP头	扩展头（可选）	AH	原始IP头	扩展头（可选）	传输层协议头	传输层协议数据

图2-5　隧道模式下AH在IPv4和IPv6中的应用

和在传送模式下一样，AH验证整个IP数据报，包括新的IP头。我们讨论过的AH在传送模式下的局限性在隧道模式下同样存在。如果希望对位于NAT环境中的通信双方进行AH隧道模式认证的话，那么验证应该在网关上而不是在通信主机上进行。

2.3.2　ESP

ESP（Encapsulating Security Payload）提供数据保密、数据源认证、无连接完整性、抗重播服务和有限的数据流保密。除了AH提供的服务外，ESP增加了两个功能：数据保密和有限的数据流保密服务。保密服务通过使用密码算法加密IP数据报和相关部分实现。数据流保密由隧道模式下的保密服务提供。

2.3.2.1　ESP 头格式[2]

ESP由4个固定长度的域和3个变长域组成。包格式如图2-6所示：

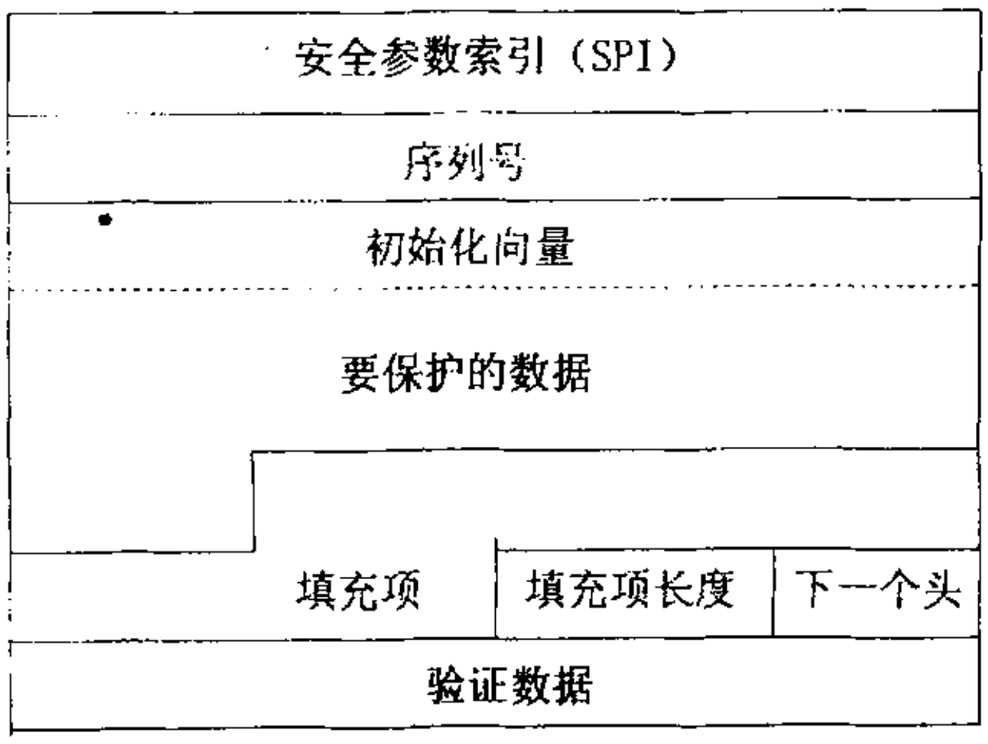


图2-6　ESP头格式

其中各域的含义如下：

1、安全参数索引、序列号和下一个头域的意义与AH类似。

2、要保护的数据：可变长。如果使用保密服务的话，其中包含实际的载荷数据（就是说，数据报加密部分的密文）。不管涉及的SA是否需要保密服务，这个域都必须有。如果加密的算法需要初始化向量（IV），它将在载荷域中传输，且算法的规范要指明IV的长度和它在载荷数据域中的位置。IV用于某种操作模式的分组密码以确保前一部分相似的明文（如IP数据报头）产生不同的密文。

3、填充：包含填充比特，由加密算法使用或用于使填充长度域和4字节字中的第3个字节对齐。

4、填充长度：8bit，填充字节的长度。

5、下一个头：8bit，表明载荷中封装的数据类型。可能是一个IPv6扩展头或传输层协议。

6、认证数据：可变长。存放ICV，它是对除认证数据域外的ESP包进行计算获得的。这个域的实际长度取决于使用的认证算法。如使用HMAC-MD5则认证数据域为128bit,如使用HMAC-RIPEMD-160则为

160bit。这个域是可选的，当指定的SA要求ESP认证服务时才包含它。

2.3.2.2　ESP 算法

ESP中使用对称密钥密码算法加密数据报。使用消息认证码（MAC）提供认证服务。对于加密和认证的算法选择因IPSec的实现不同而不同，但为了保证互操作性，RFC2406中规定了每个IPSec要强制实现的算法：加密算法是CBC模式的DES和空加密算法，认证算法是HMAC-MD5、HMAC-SHA-1和空认证算法。空加密和空认证算法分别是不加密和不认证选项。空加密和空认证算法是强制实现的，因为ESP加密和认证服务是可选的。但是，空加密和空认证算法不能同时使用。即是说ESP不能即不加密也不认证。DES是个弱加密算法，很少用于VPN解决方案。可以用AES、3DES或IDEA代替。

2.3.2.3　ESP 模式[2]

同AH情况一样,ESP也有传送模式和隧道模式两种操作模式。

传送模式下，ESP被插在IP头和所有选项后，但在传输层协议之前，或者在已应用的任意IPSec协议之前。如图2-7所示：

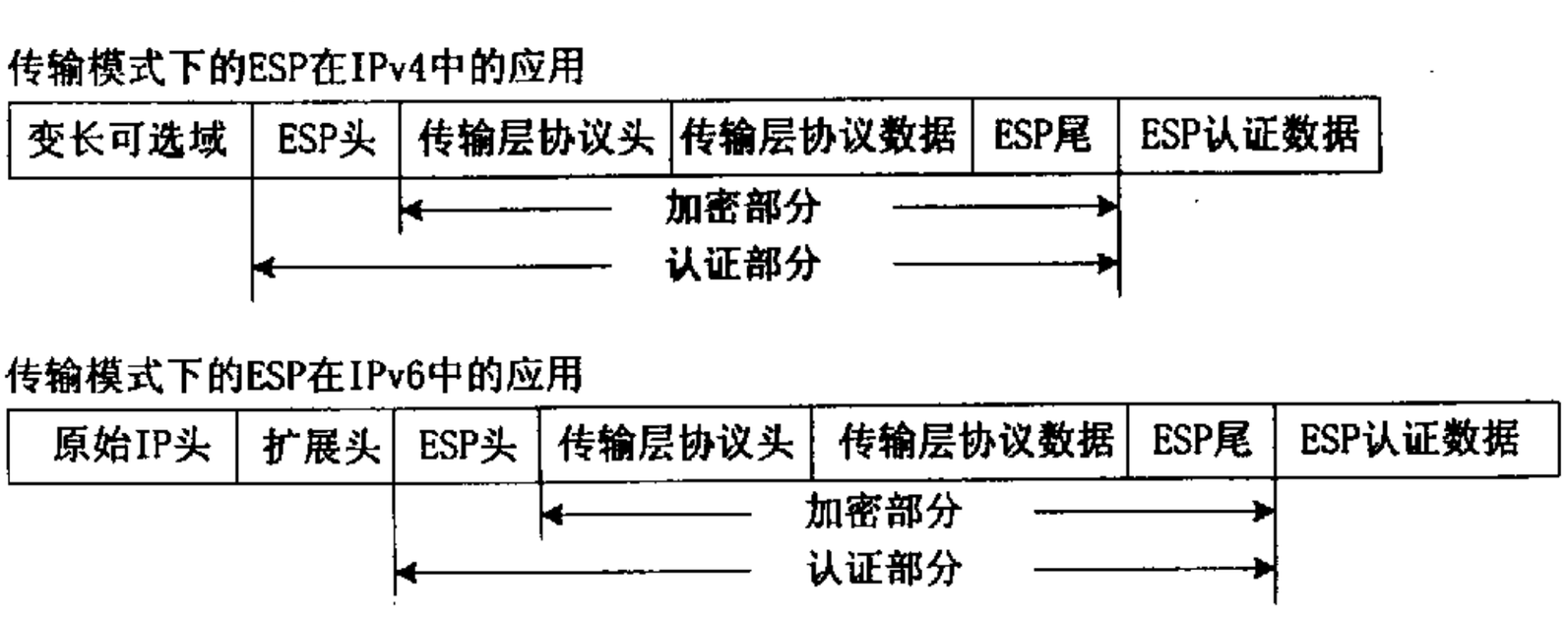


图2-7　传送模式下ESP在IPv4和IPv6下的应用

图中ESP头部域由SPI和序列号域组成，ESP尾部由填充域、填充长度域和下一个头域组成。图中标明了数据报被加密和认证的部分。如果需

要保密服务，SPI和序列号域不被加密，因为接收节点需要这些域来标志用来处理数据报的SA，另外如果启动了抗重播服务，它们还要被用来检验重播数据包。与此类似，如果有认证数据域，那它不被加密，因为如果某个SA需要ESP认证服务，目的主机在处理这个数据报之前首先用这个域来验证数据报的完整性。

ESP与AH不同，它不对整个IP数据报进行认证，所以传送模式下ESP不象AH那样受NAT的限制。使用私有IP地址或位于安全网关后的主机间的通信可被ESP认证服务保护，因为IP头中的源和目的以及其他域未被认证。然而，ESP的这种灵活性也导致了它的弱点。除了ESP头部外，在从源到目的的传输过程中IP头的任何域都可以被修改，如果修改后头部校验和计算正确，目的主机将无法检测到发生的修改。这样，ESP传送模式认证服务所提供的安全性就不如AH传送模式。同样，因为源和目的IP未被加密，所以传送模式下的ESP不提供数据流保密服务。

隧道模式的ESP如图2-8所示。ESP隧道模式认证和加密服务所提供的安全性要强于ESP传送模式，因为隧道模式的ESP加密和认证原始的IP头。因为包含源和目的地址的内部IP头被加密，所以隧道模式下的ESP可以提供数据流保密服务。

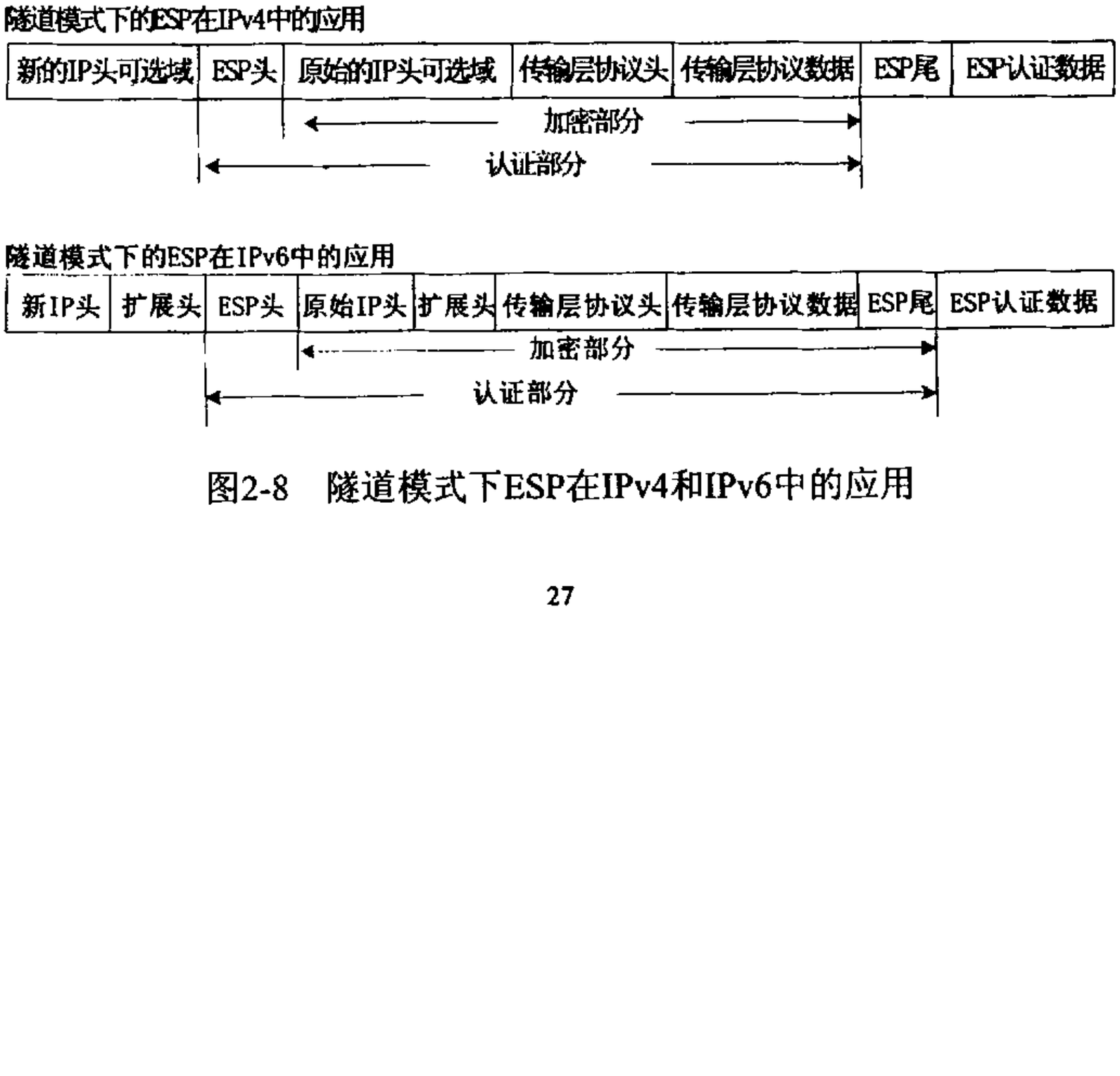


图2-8　隧道模式下ESP在IPv4和IPv6中的应用

2.3.3　AH 与 ESP 的比较[10]

AH协议和ESP协议都是网络层安全协议，但二者的侧重点不同。下面就认证、保密和防止重播三个方面加以比较。

1、认证服务：AH协议和ESP协议都提供认证服务功能。AH协议是专门用以提供认证服务；ESP协议的认证服务是它的可选项，主要为了防止对协议的“剪贴”攻击。ESP提供的认证服务范围要比AH的窄，即ESP头以前的IP报文部分不会被保护；而AH协议认证了几乎所有的IP报文字段。

2、保密服务：AH协议不提供保密服务，当不需要保密的情况下，AH协议是一种恰当的安全协议；ESP主要用于数据保密。当使用隧道方式时，位于两个安全网关间的使用ESP保护的安全联盟还可以提供一定的流量保密性。隧道模式下，由于内层的IP包被加密，所以隐藏了报文的实际源头和终点。更进一步的是，ESP使用的填充字节隐藏了报文的实际尺寸，从而更好的隐藏了这个报文的外在特性。

3、防止重播：AH协议和ESP协议都有防止重播的功能。只要接收方决定使用此功能，AH协议的此项功能就可靠工作，而ESP协议必须要有认证机制的配合，此项功能才起作用。

在具体选择安全协议时，要根据网络安全策略和网络中可能发生的攻击途径来取舍。对于防范侦听这类被动攻击，可以选择ESP协议；对于网络控制消息的保护，如ICMP、RIP，因为主要为防止假冒，故可用AH协议；对于IP伪装，如源选择攻击，也可以用AH协议；对于会话拦截，则可以使用AH协议认证TCP的连接序列号，也可以用ESP协议加密会话连接。

2.4 Internet密钥交换

IPSec保护一个包之前，必须先建立一个安全联盟（SA），SA可以手工创建或动态建立。Internet密钥交换（IKE）就是用于动态建立SA。IKE代表IPSec对SA进行协商，并对SADB数据库进行填充。IKE属于一种混合型协议，它使用到了三个不同协议的相关部分：沿用了Internet安全联盟和密钥管理协议（ISAKMP）的基础，Oakley的模式和SKEME的共享和密钥更新技术，从而定义出自己独一无二的验证加密材料生成技术，以及协商共享策略。

2.4.1 ISAKMP

ISAKMP定义了协商、建立、修改和删除SA的过程和包格式。它提供了一个通用的SA属性格式框架和一些可由不同密钥交换协议使用的协商、修改、删除SA的方法。ISAKMP是一个与密钥交换无关的协议。不受限于某个具体的密钥交换协议、密码算法、密钥生成技术或认证机制。ISAKMP以认证和受保护的形式为网络互联单元提供服务。通信双方可以向对方提供自己支持的功能从而协商共同的安全属性。ISAKMP的消息可以由TCP和UDP传输。TCP和UDP的端口500由IANA为ISAKPM保留。ISAKMP双方交换的信息以载荷的形式传输。目前已定义13种载荷。

ISAKMP提供两个协商阶段：阶段1和阶段2。每个阶段的协商都使用ISAKMP定义的交换或其他交换协议（如IKE）定义的交换来完成。

阶段1协商：ISAKMP通信双方建立一个ISAKMP SA，这个SA时关于如何保护双方以后的协商通信的一致意见。然后使用这个SA来保护其他协议SA的协商。

阶段2协商：这个阶段用于建立其他安全服务的SA，例如ESP或AH。一个单独的阶段1的SA可以用于建立很多阶段2的SA。

2.4.2　IKE

ISAKMP提供了一个可由任意密钥交换协议使用的通用密钥交换框架，但是它本身没有定义具体的密钥交换技术。密钥交换的定义留给其他协议处理。对IPSec而言，已定义的密钥交换就是IKE—即Internet密钥交换。IKE利用ISAKMP语言来定义密钥交换，是对安全服务进行协商的手段。事实上，IKE定义了为数众多的交换方式，以及相关的选项。IKE交换的最终结果是一个通过验证的密钥以及建立在双方同意基础上的安全服务—亦即所谓的“IPSec安全联盟（IPSec SA）”。

IKE使用了两个阶段的ISAKMP。第一阶段建立IKE安全联盟，第二阶段利用这个既定的安全联盟，为IPSec协商具体的安全联盟。和ISAKMP不一样，IKE为其安全联盟的属性进行了定义。但尽管如此，还是没有定义其他任何安全联盟的属性。这种定义留待解释域（DOI）进行。DOI规定了IKE在阶段2交换中需要协商解决的可选及必需属性。尽管IKE没有定义自己的DOI，但却围绕安全联盟的使用，规定了相关的条款。

IKE定义了两个阶段1交换：主模式交换和野蛮模式交换，它们都通过短暂的Diffie-Hellman交换来产生经过验证的密钥材料，主模式必须要实现，野蛮模式的实现是可选的；另外，它还定义了一个阶段2交换：快速模式交换。主模式交换在三个步骤中采用了六条消息，最终建立了IKE SA。这三个步骤分别是模式协商、一次Diffie-Hellman交换和一次nonce交换、以及对对方身份的验证。主模式的特点包括身份保护和对ISAKMP协商能力的完全利用。身份保护在对方希望隐藏自己的身份时

显得十分重要。野蛮模式交换的用途与主模式交换相同—建立一个验证的安全联盟和密钥，随后可用IKE为其他安全协议建立安全联盟。主要的差别在于，野蛮模式只需用到主模式一半的消息。由于对消息的数量进行了限制，野蛮模式同时也限制了它的协商能力，而且不会提供身份保护。建立好IKE SA之后，可用它为其他完全协议（如IPSec）生成相应的SA。这些SA是通过快速模式交换来建立的，对一次快速模式交换来说，它是在以前建立好的IKE SA的保护下完成的。IKE SA保护快速模式交换的方法是：对其进行加密，并对消息进行验证。在一次快速交换模式中，通信双方需要协商拟定IPSec安全联盟的各项特征，并为其生成密钥。

2.5 PF_KEY[12]

PF_KEY 由 PF_ROUTE（BSD routing socket） 衍生而来，是一个新的套接口协议族，用于可信赖的有特权的密钥管理程序和操作系统内部的密钥管理(这里指“密钥引擎”或者安全联盟数据库)的通信。在协议家族中，PF_KEY 被定义为 15。我们采用的是 PF_KEY 第 2 版本。PF_KEY 与其它 socket 不一样，它不使用任何 socket 地址。PF_KEY 域当前只支持 SOCK_RAW 类型，协议字段必须设为 PF_KEY_V2。

大多数密钥管理通常部分或者全部在应用层实现。（例如：IPsec 的密钥管理提案 ISAKMP/Oakley 等都是应用层协议。）图 2-9 说明了密钥管理守护进程和 PF_KEY 之间的关系。密钥管理守护进程使用 PF_KEY 与密钥引擎或安全联盟数据库（SAD）通信，并且使用 PF_INET（在 IPv6 下用 PF_INET6）通过网络与远端的密钥管理程序通信。

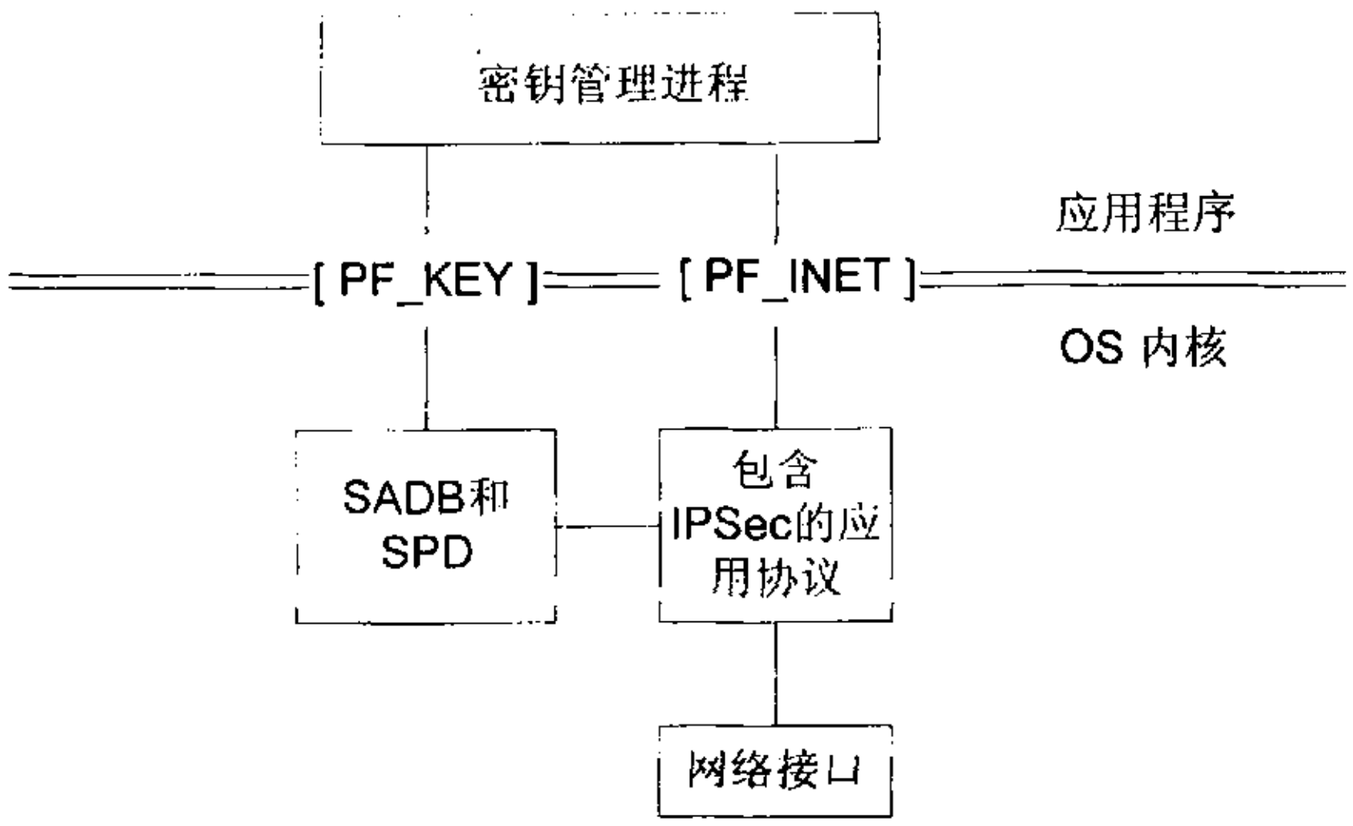


图 2-9 密钥管理守护进程与 PF_KEY

一个进程使用 PF_KEY 套接字发送和接受消息与密钥引擎相互作用，通过一系列预先定义的消息，安全联盟信息可以被插入到内核的安全联盟表并被检索。在正常情况下，所有正确的消息发送至内核并被返回到包括发送者在内的所有打开的 PF_KEY 套接字。格式错误的消息将会造成错误，对应一个具体的实现，它必须在返回消息给合适的监听者之前检查消息的正确格式。

2.5.1　消息格式

PF_KEY 消息由一个基本的消息头和一些可选的附加数据段组成，附加的数据段格式由消息类型决定。

2.5.1.1　基本消息头

基本消息头使用 POSIX 类型，格式如图 2-10。

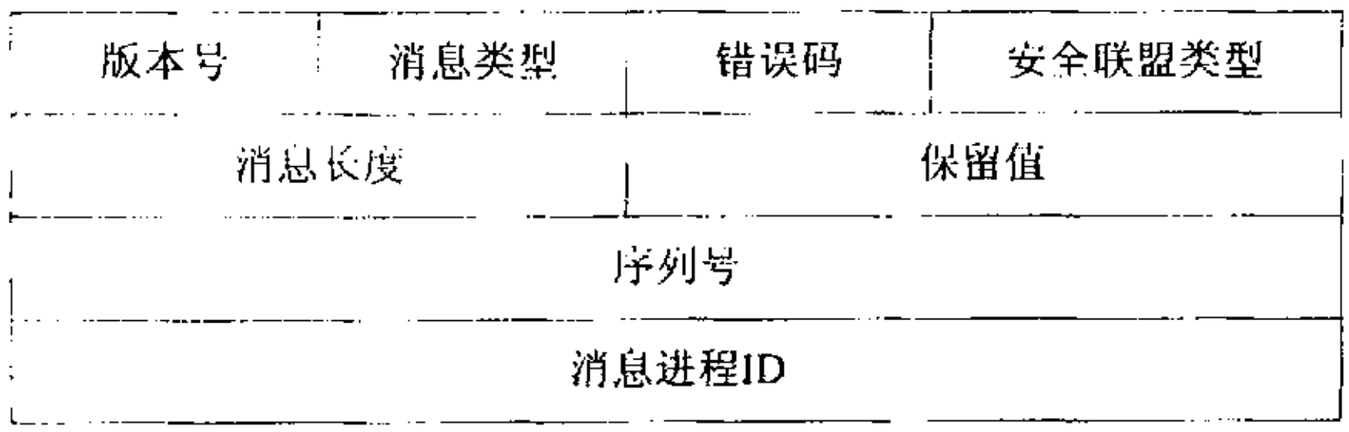


图 2-10 PF_KEY 基本消息头格式

其中各域意义如下：

1、版本号：PF_KEY 消息的版本号，必须设置为 PF_KEY_V2。否则，write()调用将失败并报参数错误（EINVAL）。另外，应用程序不能理解从内核收到的消息的格式，运行状态将无法确定。

2、消息类型：标识 PF_KEY 消息的类型。

3、错误码：消息的发送者应设置此项为零。如果出现错误，消息的应答在此存放错误码。包括应用层的消息应答。（如：应用层协商失败，将返回错误号）

4、安全联盟类型：标识安全联盟的类型。有效的安全联盟类型在<net/pfkeyv2.h>中定义。

5、消息长度：包含消息的整个长度，64 位字对齐，包括基本头长度和附加数据，并包括可能存在的任何填充和额外空间。除非其它情况，所有其它长度字段也是 64 位字对齐。用户到内核的消息，这一项必须检验。如果检验失败必须返回错误 EMSGSIZE。内核到用户的消息，大小不匹配就像用户未提供足够的缓存。在这种情况下，用户程序可以丢弃这个消息，但是应努力析取超出的数据。

6、保留值：消息发送者必须设置为零。本文档中所有的保留值含义与此相同。

7、序列号：包含消息的序列号。这一项必须连同 sadb_msg_pid 唯一确定一个进程的请求。发送者负责设置这一项并负责匹配一个请求的包含消息的序列号（如：SADB_ACQUIRE）。这一项就像一个同远端程序调用实现的交易 ID。

8、消息进程 ID：识别消息的发起进程或者消息的目的进程。例如：如果 ID 为 2112 的进程发送一个 SADB_UPDATE 消息至内核，必须设置本项为 2112，并且内核在自己的应答消息中设置本项为 2112。本项连同序列号能够唯一确定一个进程的请求。通常用一个 32 位值保存操作系统的进程 ID。

2.5.1.2 扩展消息头

基本消息头后的附加数据由一系列长度-类型值字段组成。开始的 32 比特固定格式如下：

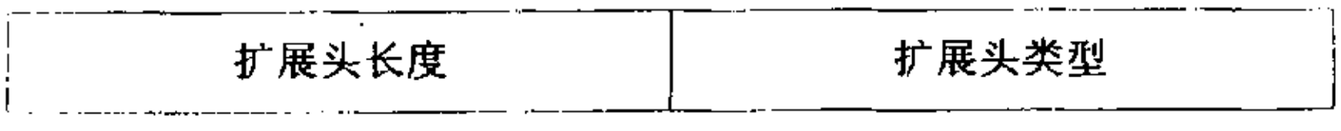


图 2-11　PF_KEY 扩展消息头格式

其中：

1、扩展头长度：扩展头 64 位字对齐的长度。

2、扩展头类型：扩展项的类型如下，零为保留值。

结构体标识	扩展项类型	解释
sadb_sa	安全联盟扩展项	说明了单个安全联盟的详细数据
sadb_lifetime	生存期扩展项	说明了安全联盟的一个或多个生存周期变量
sadb_address	地址扩展项	说明一个或多个与安全联盟有关的地址

sadb_key	密钥扩展项	说明了一个或多个与安全联盟有关的密钥
sadb_ident	身份扩展项	包含了端点的身份特征
sadb_sens	敏感度扩展	包含安全联盟的安全标签信息
sadb_prop	提议扩展项	包含了可选择的算法参数
sadb_supported	支持算法扩展项	包含系统所支持的算法列表，密钥管理程序可以协商使用
sadb_spirange	SPI 范围扩展项	SADB_GETSPI 消息需要一个可接受的 SPI 范围

表 2-1　PF_KEY 扩展项

2.5.2　消息机制

PF_KEY 定义了一套自己的消息行为，用以完成自己的消息通信机制，各种消息行为如下所示[12]：

1、SADB_REGISTER 消息：进程向内核注册一个 socket 以便能够从内核获得新的 SA。消息格式为：<base>，即只包含消息基本头。内核将发送 SADB_REGISTER 消息进行应答，格式为：<base,supported>，supported 扩展表明内核所支持的验证算法和密钥算法。

2、SADB_ACQUIRE 消息：这个消息是由内核发出的，可以用于两种情况：

1) 内核向密钥管理进程申请创建一个 SA。消息格式<base,address,proposal>，其中，proposal 扩展给出了 SA 的几种提案，即采用什么协议（AH 或 ESP）、采用什么验证方法和加密方法等；扩展数据中还可包含的选项有：Identity 扩展，Sensitivity 扩展。但密钥管理进程并不向内核发送

SADB_ACQUIRE 应答消息。

2) 当密钥协商过程失败后，内核通过发送 SADB_ACQUIRE 消息来通知密钥管理进程。消息格式为：<base>，即只包含一个消息基本头，其中，消息错误号代表了错误的原因。密钥管理进程不向内核发送 SADB_ACQUIRE 应答消息。

3、SADB_GETSPI 消息：密钥管理进程向内核申请一个 SPI。消息格式为：<base,address,SPI range>，SPI range 扩展指出了所申请的 SPI 的范围。内核发送 SADB_GETSPI 应答消息，格式为：<base,SA(*),address>，其中，SA(*)表示 SA 扩展的各个字段，除了扩展长度、扩展类型和 SPI 外，其余字段应设为 0 被忽略掉。

4、SADB_UPDATE 消息：密钥管理进程请求内核更新（或）增加一条 SA 记录，该消息随 SADB_GETSPI 消息一起使用。消息格式为：<base,SA,address,key>，扩展数据中还可包含的选项有：lifetime 扩展、identity 扩展、sensitivity 扩展。内核发送 SADB_UPDATE 应答消息，格式为：<base,SA,address>,或者还包含与请求消息相应的扩展数据。

5、SADB_ADD 消息：在手工配置密钥或知道 SPI 的情况下，无须先使用 GETSPI 消息而直接申请内核更新（或增加）一条 SA 记录。消息格式与 SADB_UPDATE 消息相同。内核发送的 SADB_ADD 应答消息与 SADB_UPDATE 应答消息相同。

6、SADB_DELETE 消息：密钥管理进程请求内核删除一条 SA 记录。消息格式为：<base,SA(*),address>，即根据 SA 类型、SPI、地址三要素删除相应的 SA。内核发送 SADB_DELETE 应答消息为：<base,SA(*),address>。

7、SADB_GET 消息：密钥管理进程请求从内核检索一条 SA 记录。消息格式为：<base,SA(*),address>。内核发送 SADB_GET 应答消息为：

<base,SA,address,key>，扩展数据中还可包含的选项有：lifetime 扩展、identity 扩展、sensitivity 扩展，即应答消息给出 SA 的详细信息。

8、SADB_EXPIRE 消息：内核向密钥管理进程发送 SA 过期消息。消息格式为：<base,SA,lifetime,address>，lifetime 扩展给出了 SA 的当前时间和过期时间（软过期和硬过期）。密钥管理进程不向内核发送 SADB_EXPIRE 应答消息。

9、SADB_FLUSH 消息：密钥管理进程请求内核将某种类型的 SA 从 SADB 中全部删除。消息格式为：<base>，消息基本头中 SA 类型字段给出了所要删除的 SA 类型的值。内核发送 SADB_REGISTER 应答消息，格式为：<base>。

10、　SADB_DUMP 消息：一般用于 PF_KEY 实现的调试。

2.5.3　工作原理

下面介绍使用 PF_KEY 的基本原理[13]：

当密钥管理守护进程（KMd）开始，必须告诉 PF_KEY 将要接收消息用于 IPSec 的两个服务，AH 和 ESP。通过发送两个 SADB_REGISTER 消息完成这些。

KMd->Kernel: SADB_REGISTER for ESP

Kernel->Registered: SADB_REGISTER for ESP, Supported Algorithms

KMd->Kernel: SADB_REGISTER for AH

Kernel->Registered: SADB_REGISTER for AH, Supported Algorithms

每一个 REGISTER 消息将导致一个应答给所有分别注册到 ESP 和 AH 的 PF_KEY 套接字（包括请求发送者）。

假设 IPSec 当前可使用的安全联盟不存在，认为一个网络程序开始传送数据（例如一个 TCP 的 SYN）。因为策略关系，或者程序的要求，内核 IPSec 模块需要一个 AH 安全联盟用于这些数据。由于不存在，产生以下消息：

Kernel->Registered: SADB_ACQUIRE for AH, addrs, ID, sens, proposals

KMd 读取 ACQUIRE 消息，尤其是 sadb_msg_seq 值。在开始协商前，发送含有相同 sadb_msg_seq 值的 SADB_GETSPI 消息。内核返回 GETSPI 的结果给所有的监听套接字。

KMd->Kernel: SADB_GETSPI for AH, addr, SPI range

Kernel->All: SADB_GETSPI for AH, assoc, addrs

KMd 可以完成第二个 GETSPI 操作，如果它需要双向的 IPSec SPI 值，KMd 得到 SPI 后，开始协商。在派生出密钥素材，并协商其它参数后，发送一个（或多个）包含同一 sadb_msg_seq 值的 SADB_UPDATE 消息。

如果在协商中 KMd 出现任何错误，将发送：

KMd->Kernel: SADB_ACQUIRE for AH, assoc (with an error)

Kernel->All: SADB_ACQUIRE for AH, assoc (same error)

如果成功，将发送：

KMd->Kernel: SADB_UPDATE for AH, assoc, addrs, keys, <etc.>

Kernel->All: SADB_UPDATE for AH, assoc, addrs, <etc.>

UPDATE 的结果（去掉实际的密钥）发送给所有的监听套接字。如果只有 SPI 值在本地确定，其它 SPI（由于 IPSec 的 SA 是单向的）必须用 SADB_ADD 增加。

KMd->Kernel: SADB_ADD for AH, assoc, addrs, keys, <etc.>

Kernel->All:　SADB_ADD for AH, assoc, addrs, <etc.>

如果扩展项中有生存期扩展，当生存期中的一项期满，将收到 SADB_EXPIRE 消息。

Kernel->All: SADB_EXPIRE for AH, assoc, addrs, Hard or Soft,

Current, <etc.>

KMd 可以使用以这个消息为线索开始协商，或者，如果对策略有发言权，发送带生存期扩展的 SADB_UPDATE 消息。

第三章 嵌入式实时操作系统 VxWorks

3.1 嵌入式实时操作系统概述

一般来说，嵌入式系统可以被定义为：以某种特定功能的应用为中心、以计算机技术为基础、软件硬件紧密结合并可裁减、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。具有软件代码小，高度自动化，响应速度快等特点，特别适合于要求实时的和多任务的应用体系。

嵌入式实时操作系统是运行在嵌入式系统中的实时操作系统，跟通用操作系统相比，实时操作系统确保了任务运行的实时性、可确定性、可靠性。

嵌入式系统中引入操作系统，使得嵌入式开发和个人计算机、小型机等之间在开发上的差别正在逐渐消除，软件工程中的很多经验、方法乃至库函数可以移值到嵌入式系统。不同嵌入式系统中的软件也很方便移值。优秀的嵌入式操作系统上的用户程序，在跨平台移值的时候只需要修改 1%到 4%的代码。

采用实时操作系统，可以充分利用操作系统所提供的高可靠性和高实时性，大大简化了开发过程。

3.2 VxWorks 的构成和特点[8]

VxWorks 是美国 WindRiver 公司于 1983 年设计开发的一种嵌入式实时操作系统（RTOS），它以其良好的持续发展能力、高性能的内核以及友好的用户开发环境，在嵌入式实时操作系统领域中占据一席之地。

VxWorks 操作系统包括了进程管理、存储管理、设备管理、文件系统管理、网络协议及系统应用等几个部分。VxWorks 只占用了很小的存储空间，并可高度裁减，保证了系统能以较高的效率运行。它的体系结构如下图 3-1 所示：

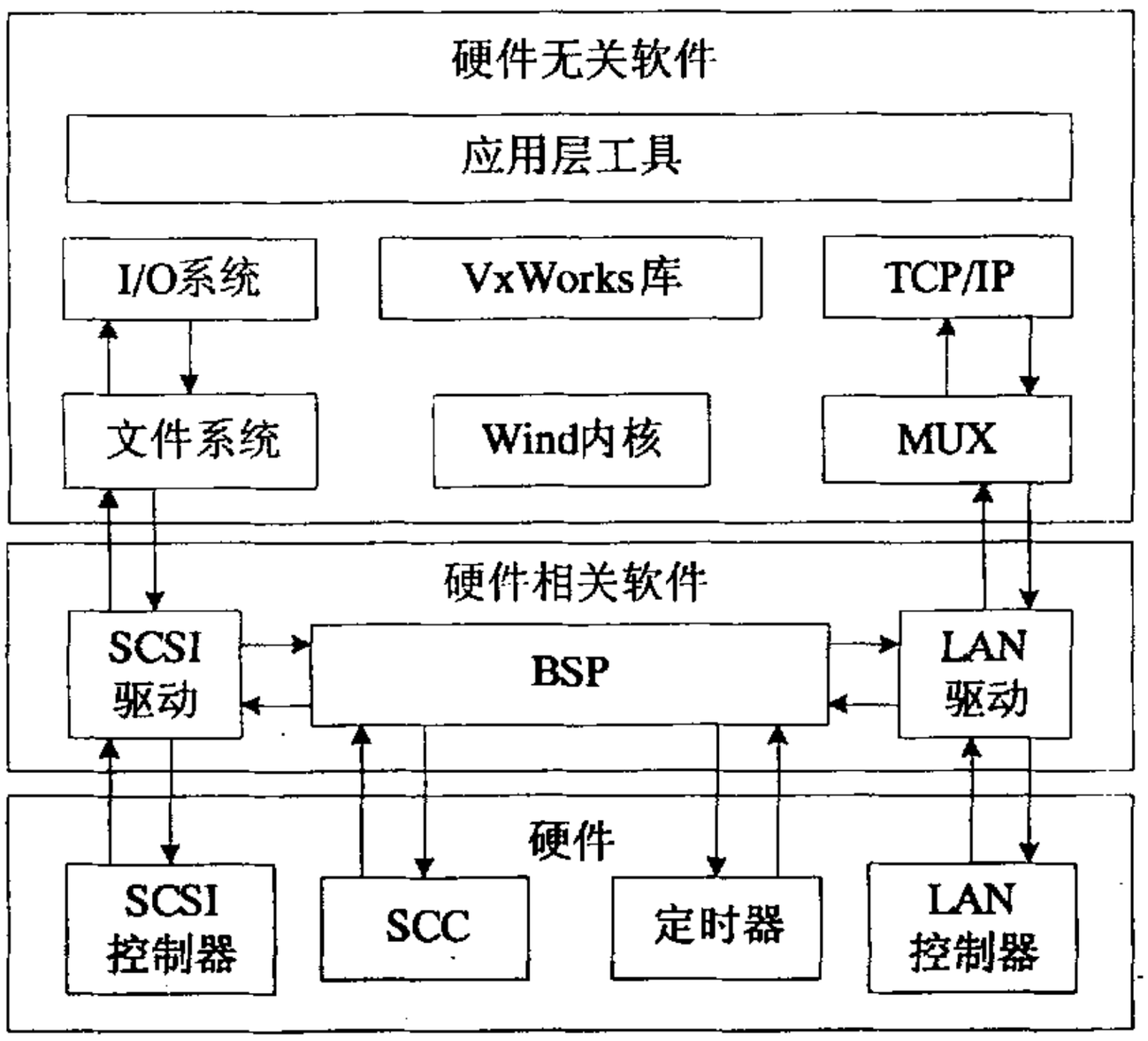


图 3-1 VxWorks 体系结构图

1、高性能实时操作系统核心－Wind Kernel

VxWorks 的系统核心是微内核结构，它提供的功能包括多任务调度，任务间的同步和进程间通信机制以及中断处理、看门狗和内存管理机制。一个多任务环境允许实时应用程序以一套独立任务的方式构筑，每个任务拥有独立的执行线程和它自己的一套系统资源。这些任务基于进程间通信机制同步、协调其行为。

2、I/O 系统

VxWorks 提供了一个快速灵活的与 ANSIC 兼容的 I/O 系统，包括

UNIX 标准的缓冲 I/O 和 POSIX 标准的异步 I/O。

3、实用库

VxWorks 提供了一个实用例程的扩展集，包括中断处理、看门狗计时器、消息登录、内存分配、字符扫描、线缓冲和环缓冲管理、链表管理和 ANSIC 标准。

4、网络设施

VxWorks 提供了对其他网络和 TCP/IP 网络系统的“透明”访问，包括与 BSD 套接字兼容的编程接口，远程过程调用(RPC)，SNMP(可选项)，远程文件访问以及 BOOTP 和 ARP 代理。所有的 VxWorks 网络机制都遵循标准的 Internet 协议。

5、文件系统

VxWorks 支持四种文件系统：dosFs，rt11Fs，rawFs 和 tapeFs。普通数据文件，外部设备都统一作为文件处理。它们在用户前没有相同的语法定义，使用相同的保护机制。

6、硬盘驱动

VxWorks 包括以下驱动程序：网络驱动、管道驱动、RAM 盘驱动、SCSI 驱动等等。

7、板级支持包 BSP（Board Support Package）

板级支持包对各种板子的硬件功能提供了统一的软件接口，它包括硬件初始化、中断的产生和处理、硬件时钟和计时器管理、局域和总线内存地址映射、内存分配等等。每个板级支持包括一个 ROM 启动(BootROM)或其他启动机制。

总体说来，VxWorks 有以下特点：

1、可裁剪微内核结构，对外设不作假设，系统可移值性好。

2、高效的任务管理。

3、多任务，具有 256 个优先级。

4、具有优先级排队和循环调度。

5、快速的、确定性的上下文切换。

6、灵活的任务间通讯：信号灯、消息队列、套接字、共享内存。

7、微秒级的中断处理。

8、支持 POSIX 1003.1b 实时扩展标准。

9、支持多种物理介质及标准的、完整的 TCP/IP 网络协议。

10、　灵活的引导方式：支持从 ROM、flash、本地盘或网络引导。

11、　支持多处理器并行处理。

3.3　VxWorks 系统的开发环境 Tornado

Tornado 是嵌入式实时领域里最新的开发调试环境，是实现嵌入式实时应用程序的完整的软件开发平台，是交叉开发环境运行在主机上的部分，是开发和调试 VxWorks 系统不可缺少的组成部分。Tornado 给嵌入式系统开发成员提供了一个不受目标机资源限制的超级开发和调试环境。

嵌入式系统开发一般包括硬件开发和软件开发，自主设计的这块板子通常没有软件的自开发能力，所以我们需要一台通用机来辅助开发，这台通用机可以是 PC 或工作站，我们称辅助我们软件开发的通用机为宿主机(Host)，用户自己开发的板子为目标机(Target)。宿主机上要有一个集成开发环境(IDE)来辅助我们的软件开发，这套集成开发环境可以运行在 Windows95/NT 或 UNIX 下，包括交叉编译器(Cross Compiler)和交叉调试器(Cross Debugger),所谓交叉编译器就是在宿主机上编译生成可以在目标机上运行的代码 IMAGE，交叉调试器就是通过宿主机和目标

机之间的某种耦合方式实现前后台调试。我们称宿主机上的这套集成开发环境为 Tornado，编译生成的目标机上的可执行代码 IMAGE 为 VxWorks。在系统安装的时候，集成调试环境和 VxWorks 的原材料（一些 obj 文件）都安装到宿主机上，编译生成的在目标机上运行的 IMAGE 内包含操作系统。

Tornado 开发系统包含三个高度集成的部分：

1、运行在宿主机和目标机上的强有力的交叉开发工具和实用程序。

2、运行在目标机上的高性能，可裁剪的实时操作系统 VxWorks。

3、连接宿主机和目标机的多种通讯方式，如：以太网，串口线，ICE 或 ROM 仿真器等。

图 3-2 表明了 Tornado 开发环境的基本模型。

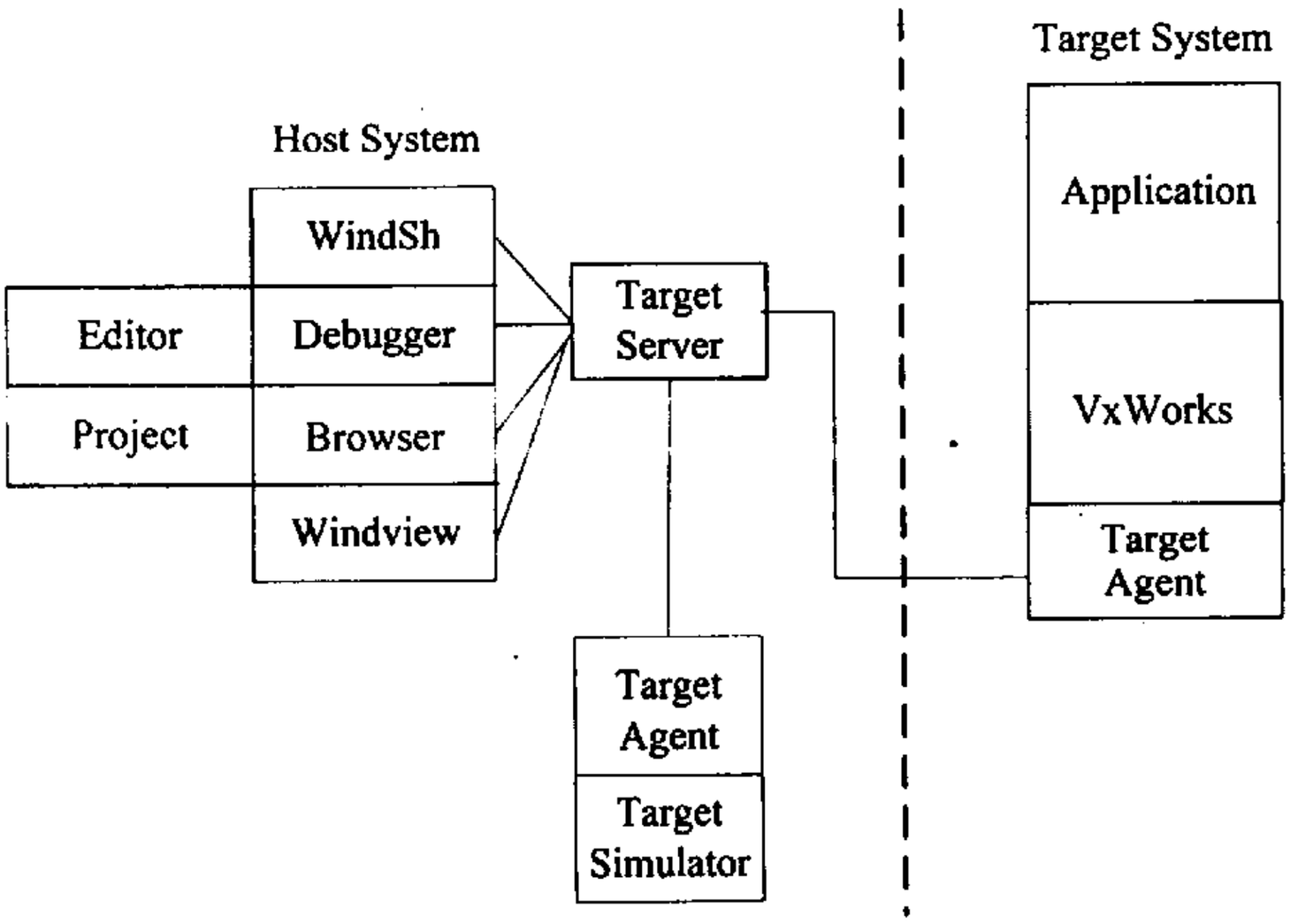


图 3-2 宿主机和目标机之间的通信[15]

从图中可以看出，主机与目标机之间的通信是通过运行各自处理器上的代理进程来完成的，这使得主机上的开发工具和目标机上的操作系

统可以完全脱离相互之间的连接。

有两个主要部件来完成宿主机和目标机的通信：

1、Target Agent：它运行在目标机上，VxWorks 通过它与主机通信。Target Agent 是 Tornado 工具在目标机上的代理。它负责接收和处理由 Target Server 发送来的各种请求，包括内存处理、事件、虚拟 I/O 和任务控制等。Agent 通常独立于所运行的操作系统。它最主要的作用是为主机驻留工作目标模块加载器服务，为 Tornado 提供了增量加载的能力。这样就大大的减少了下载模块的时间。

2、Target Server：目标机服务器，Tornado 工具通过它与 Agent 连接。每个目标机和一个 Target Server 对应，所有的主机工具通过该 Target Server 访问对应的目标机。Target Server 将工具发出的请求转换成与 Agent 对应的一系列事务处理过程。Target Server 负责管理和处理与 Agent 的连接细节。它主要提供以下服务：Ethernet 网卡、串口、NetROM 和定制连接的通信；驻留在目标机上的目标机符号表；动态模块装载和卸载；目标机的虚拟 I/O 路径。

对于不同的目标机，Tornado 给开发者提供一个一致的图形接口和人机界面。当使用 Tornado 的开发人员转向新的目标机时，不必再花费时间学习或适应新的工具；对深嵌入式应用开发者来说更重要的是，Tornado 所有的工具都是驻留在开发平台上的。在嵌入式系统工具发展历史上，Tornado 是第一个实现了当目标机资源有限时开发工具仍可使用而且功能齐全的开发环境。

Tornado 工具集包括：

1、WindSh：提供一个从宿主机到目标机之间的一个命令 shell。

2、Launcher：Tornado 的框架环境、管理目标机与工具。

3、Browser：详细查看任何目标系统对象、内存、堆栈和 CPU 占用

率。

4、WindConfigTM：图形化自动配置工具。

5、CrossWind：系统和任务级调试工具，可以调试 C、C++以及汇编程序。

6、基于 Internet 的技术支持工具。

7、适用于所有目标机结构的编译器。

8、动态下载器：对象模块的动态链接和加载及卸载。

9、WindShTM：与目标机进行交互的命令（C，TCL 语言）解释工具。

10、　　支持 C 和 C++。

第四章　路由器软件总体设计

4.1　路由器综述

随着计算机网络规模的不断扩大，大型互联网络（如 Internet）的迅猛发展，路由技术在网络技术中已逐渐成为关键部分，路由器也随之成为最重要的网络设备。路由器（Router）是用于连接多个逻辑网络的网络通讯设备，所谓逻辑网络是代表一个单独的网络或者一个子网。当数据从一个子网传递到另一个子网时，可通过路由器来完成。

4.1.1 路由器原理和功能

顾名思义，路由器主要是通过路由的选择将不同网络联系在一起，它是一种协议相关设备。路由器从数据链路层接收到网络上的数据报文，经过校验，对正确的数据包将根据报头中的信息和路由表决定路由，然后根据结果对报头进行相关的修改，并将报文发送到相应的网段。路由器拥有一张全网一致的路由表，它可以根据路由表和所连接的负载情况为数据包的转发选择一条最佳路径。此最佳路径可能是花费最小、时延最小、可靠性最高或者是最短距离等。可见，路由器的主要功能是选径和转发数据包。另一方面，路由表还用来在源和目的节点之间提供多条路径，这给网络增加了容错性、冗余能力和负载均摊等。有些情况下，当网络发生拥赛时，路由器还提供流控。另外，由于网络安全问题的日益严峻，路由器还增加了的安全访问控制、身份验证等安全功能。

4.1.2 路由器类型

路由器根据不同的标准可以分为几类。根据通信体系结构的不同可以分为单协议路由器和多协议路由器。单协议路由器仅支持一种通信体

系结构，如 IP 路由器，IPX 路由器等；多协议路由器可同时支持不同种通信体系结构的组合，如既支持 IP 又支持 IPX 等协议的路由器。路由器还可根据性能分为高端和低端路由器。高端路由器一般放在骨干网中心路由器等地方；而低端路由器通常放在网络的末端。

目前路由器已经广泛应用于各行各业，各种不同档次的产品已经成为实现各种骨干网内部连接、骨干网间互联和骨干网与互联网互联互通业务的主力军。

4.2　路由器中软件功能设计

4.2.1　总体功能设计[17]

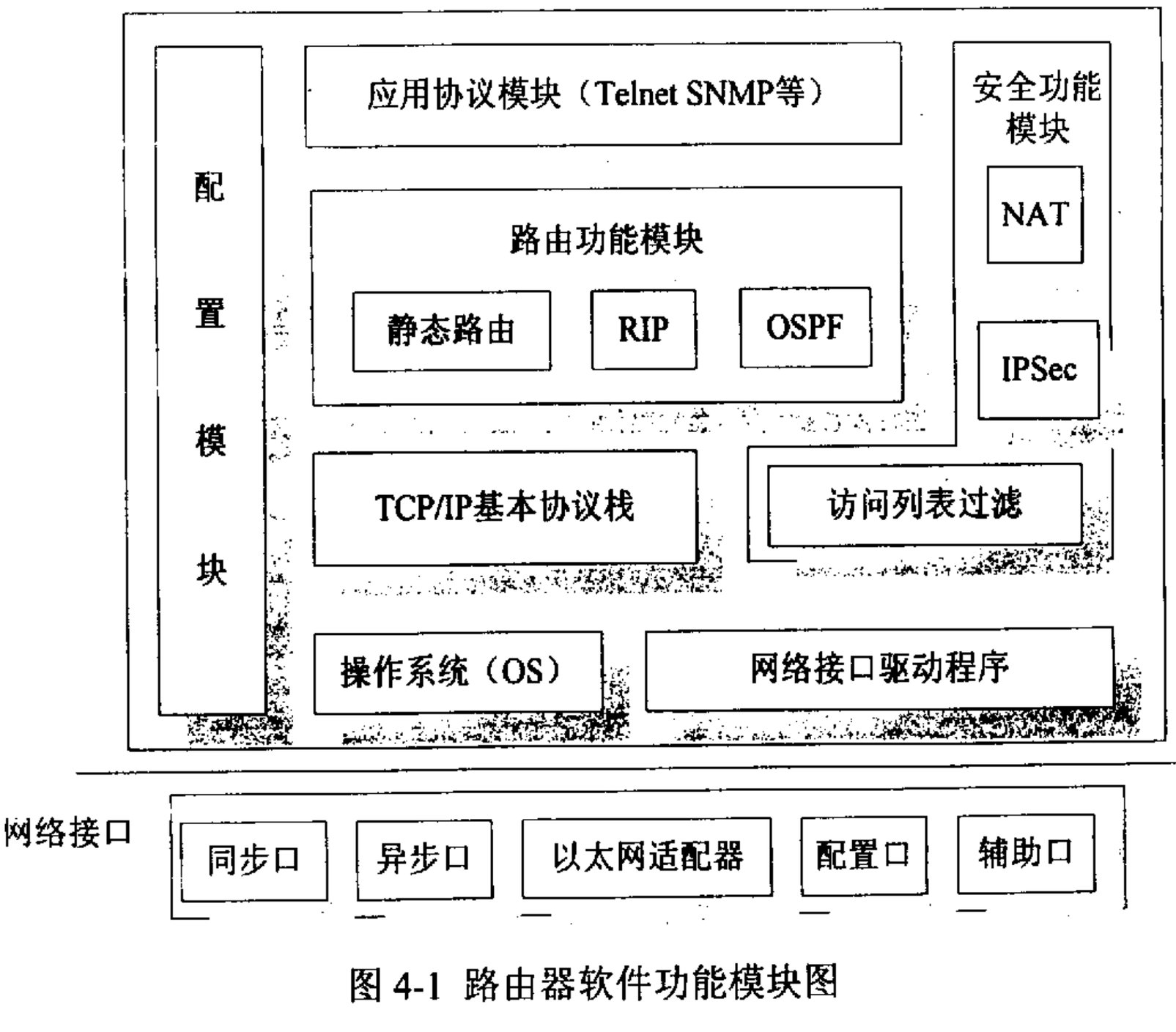


图 4-1 路由器软件功能模块图

图 4-1 概括了一般路由器软件模块的组成部分，它主要包括以下几个软件模块：

1、网络接口的驱动程序

通信网络都必然是硬件和软件的相互结合，其中软件主要指通信的协议，硬件则是数据信息传输的物理媒介，驱动程序正是提供硬件和软件接口的不可缺少的部分。在网络协议的层次结构中驱动程序完成的工作基本上应该属于物理层和数据链路层的功能，包括启动网络适配器工作、把信息分组发送到网络上或从网络上把信息分组接收下来。它是在网络协议中唯一与硬件打交道的部分，为上层协议提供可靠的接口。

2、操作系统

路由器有着非常严格的入网测试标准，对性能要求比较苛刻；另一方面，为了迅速抢占市场，又要求系统的开发在确保质量的前提下尽可能快速，可见，选择一个好的开发平台和操作系统至关重要。而嵌入式实时操作系统凭着自身高可靠性、稳定性、实时性及可裁减性可以最大限度的满足路由器对操作系统的需求。因此，路由器一般都选择嵌入式实时操作系统来作为它的软件支撑。

3、基本 TCP/IP 协议的支持

支持 TCP、UDP、ICMP、IP、ARP 等基本协议，并完全遵从相应的 RFC 文档。

4、支持多种路由协议

支持静态路由、RIP 和 OSPF 动态路由协议，并能够在多种路由协议之间协调工作。

5、支持多种复杂的应用协议，方便管理和配置

支持 SNMP（简单网络管理）、Telnet（远程登陆）、DNS（域名解析）、DHCP（动态宿主配置）等协议，从而提供丰富的管理和应用功能。

6、安全功能

1) 识别、过滤 MAC 地址、源 IP 地址、源端口号、目的 IP 地址、目的端口；识别、过滤传输协议如 TCP、UDP 等（访问列表过滤）；

2) 具有地址隐藏，内外部地址转换功能（NAT）；

3) 数据源验证，无连接数据的完整性验证，数据内容的机密性，回放保护，以及有限的数据流机密性保证（IPSec）

7、配置功能，方便用户对路由器的配置和管理

1) 通过超级终端进行配置

2) 通过 telnet 仿终端进行配置

3) 通过远程拨号进行远程配置；

并且，在终端和 telnet 上能够通过适当的命令看到各种统计和状态信息使用户对网络的性能和运行状态一目了然，也支持当前配置的保存。

8、其它功能

通过路由器的 Console 口可以方便的更新软件版本；路由的备份功能，使路由器工作更为可靠；

4.2.2 本课题所涉及的软件组成

课题的目标是要完成基于 VxWorks 和实时 Linux 两种操作系统的、能够与 IPv4 协议栈实现互连互通的、具有多操作系统移植性的“高性能 IPv6 路由器协议栈软件”。此协议栈软件主要涉及 RIP、OSPF 等路由协议，SNMP 网络管理协议和 IPSec 安全协议。这些软件协议都需要 TCP/IP 基本协议栈的支持，因此，我们也需要对 TCP/IP 基本协议栈进行必要的修改。作者主要负责 IPSec 安全协议，下面章节主要针对 IPSec 在路由器协议栈软件中的实现进行论述。

第五章　IPSec 在路由器中的设计与实现

IPSec 可在主机、网关、路由器等多种网络设施中应用。本章在研究了路由器软件功能设计的基础上，在路由器上实现 IPSec 模块，使路由器成为安全路由器。

5.1　设计思想

IPSec 在路由器中的实现方案主要有两种类型：

1、OS 集成实施：IPSec 与路由器的 OS 集成在一起。

由于 IPSec 是一个网络层协议，所以可以当作网络层的一部分来实现。IPSec 层需要 IP 层的服务来构建 IP 头。这个模型与其它网络层协议（ICMP）的实现模型是类似的。采用与 OS 集成实施的方式，使在其内核源码中增加对 IPSec 的支持，将 IPSec 集成到内核 IP 模块中，可以使它更有利于诸如分段、路由之类的网络服务。而这种集成使 IPSec 成为操作系统内核的一部分，不仅避免了相同功能的重构，还有利于保证 IPSec 的互操作性和配置上的灵活性与可伸缩性。此外，在每个数据流的级别提供安全服务更为容易，因为密钥管理、基本 IPSec 协议和网络层可以无缝集成在一起。

2、线缆中的块（BITW）：将实现了 IPSec 的设备直接接入路由器的物理接口。

对只提供 VPN 解决方案的系统来说，OS 集成方案有一个不容忽视的缺陷，就是它不得不使用 OS 厂商提供的特性。这样可能会限制提供高级方案的能力。要解决这一限制，IPSec 的实现在一个设备中进行，那个设备直接介入路由器的物理接口。该设备不运行任何路由算法，而

是只用来保障数据包的安全。但 BITW 不能作为一种长期方案来使用，因为不可能让一个设备连接路由器的每个接口。而且这个实现方案最大的问题就是功能上的重复，它要求实现大部分的网络层特性，包括分段和路由表等。当然它也有好处，它只需要一次实现，就可以提供完整的解决方案。对提供系统集成方案的厂商来说，这是很好的。

由于作者本身就参加了路由器协议栈软件整体方案的设计，基本的 TCP/IP 协议族支持也是由本课题团队自己开发，所以决定在此基础上在路由器的 IP 层中增加 IPSec 模块，即实现 IPSec 在路由器上的原始实施。

5.2　IPSec 在路由器中总体设计

从上一章路由器的软件功能设计来看，IPSec 协议可以作为一个协议模块放入路由器的网络协议栈中。报文输入时由 IP 协议模块交给 IPSec 协议模块，输出时由 IPSec 协议模块将报文交给 IP 协议模块。

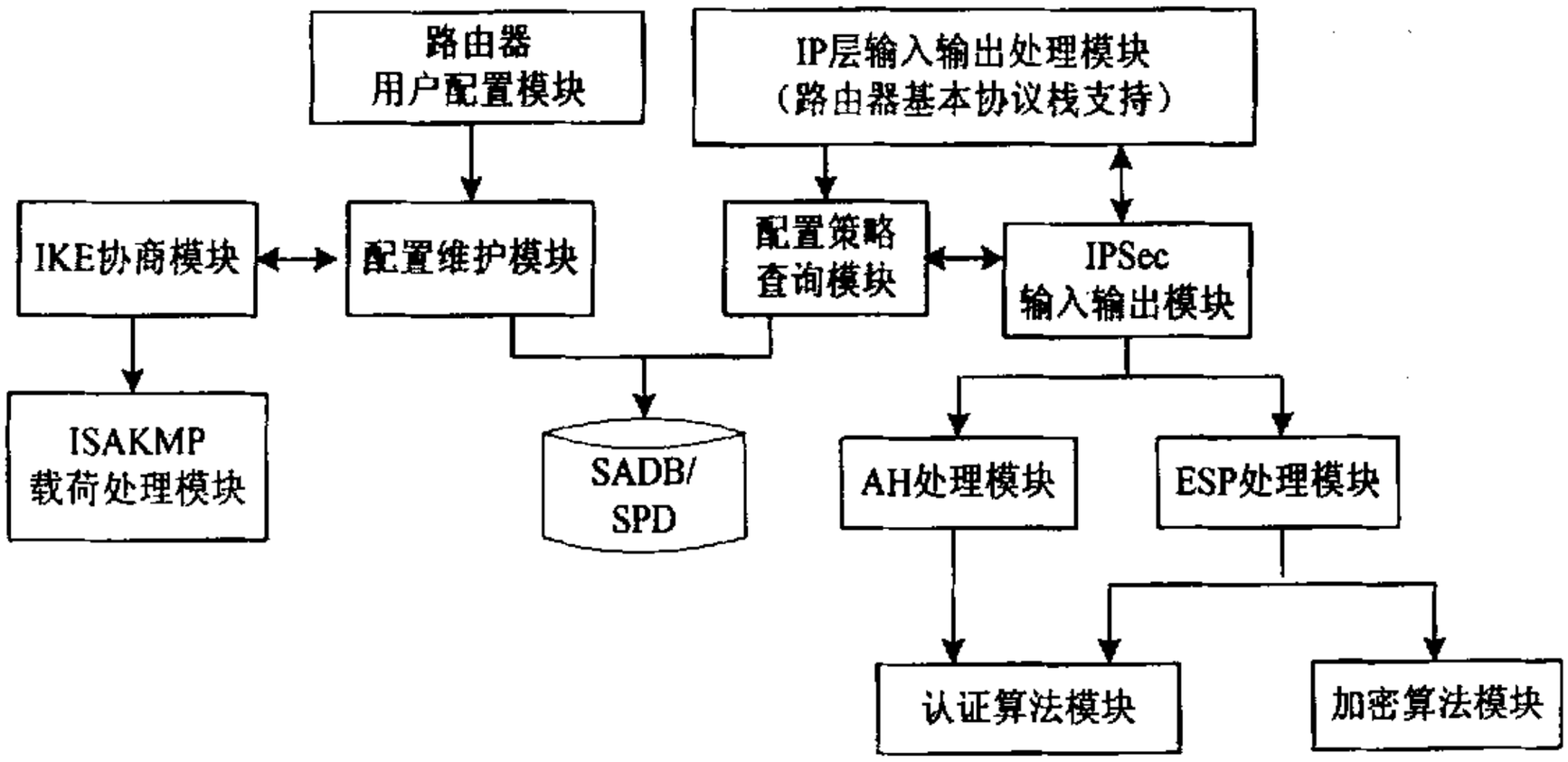


图 5-1　IPSec 在路由器中实现总体结构图

如图 5-1 所示，用户通过路由器用户配置界面可以按照自己的需求

配置安全参数，配置维护模块接收这些参数进行分析，或者直接写入数据库模块，或者是调用 IKE 协商模块，与对等端动态协商生成数据库的内容再写入，协商期间将会调用 ISAKMP 载荷处理模块。IP 层的输入输出处理模块根据对配置策略的查询，决定是否需要调用 IPSec 处理，而 IPSec 输入输出处理模块在进行 IPSec 处理的时候会调用 AH 的处理和 ESP 的处理，AH 处理模块只涉及到认证算法处理，而 ESP 处理模块则包括认证算法和加密算法的处理。

IPSec 在路由器中的工作流程如下图 5-2 所示：

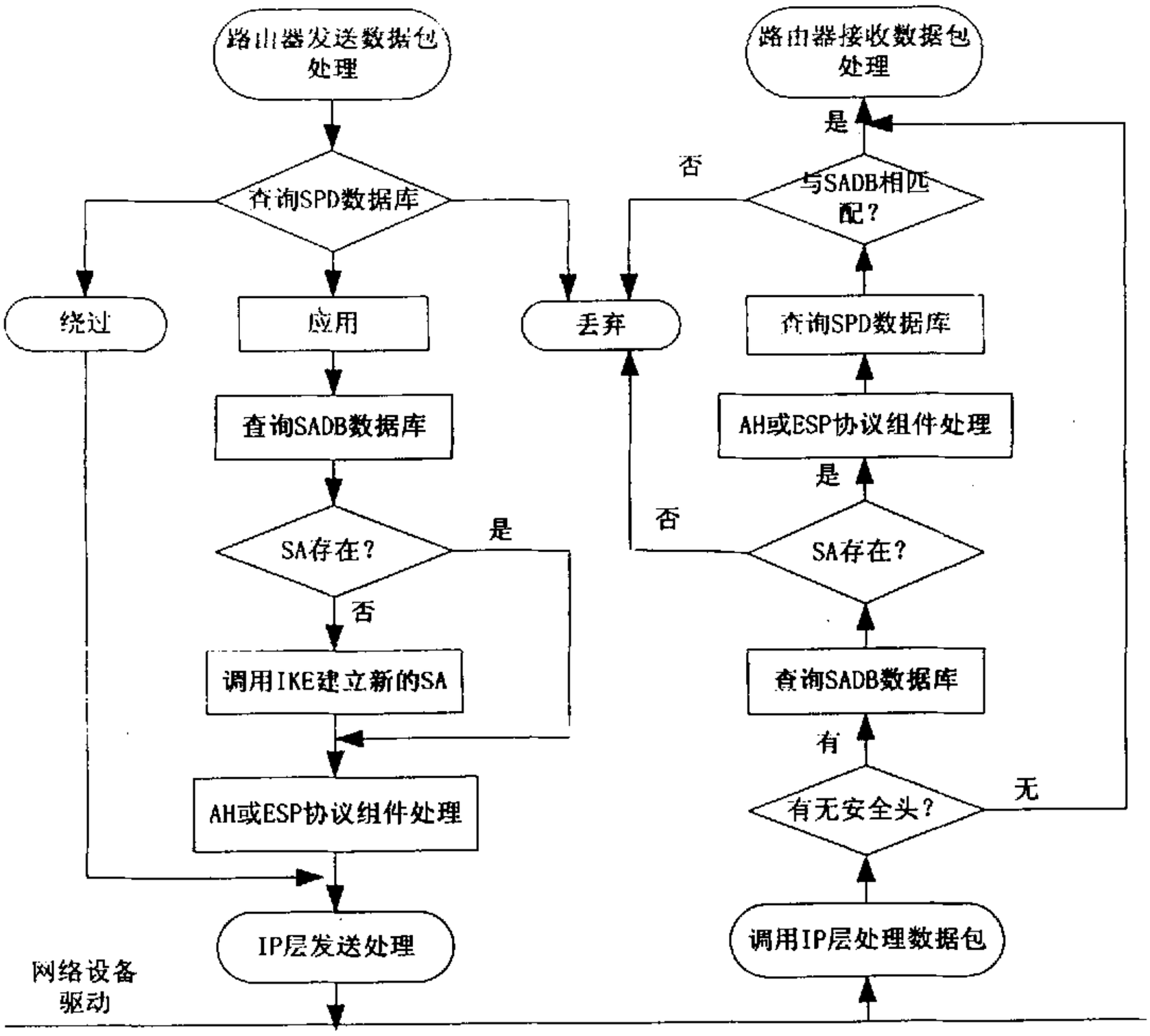


图 5-2　IPSec 的工作流程

5.3　基于 VxWorks 的 IPSec 设计实现

在路由器中我们选择 VxWorks 嵌入式实时操作系统作为路由器实施的操作系统，下面将详细描述 IPSec 在 VxWorks 网络协议栈中的实现。首先先来介绍一下 VxWorks 的网络协议栈构成。

5.3.1　VxWorks 网络协议栈

VxWorks 中的网络协议栈叫做 SENS（Scalable Enhanced Network Stack），即可裁减增强网络协议栈。SENS 是 VxWorks 内部的一个 BSD4.4 兼容的实时 TCP/IP 协议栈，它从基于 BSD4.3 的协议栈升级而来，增加了完全的路由支持以及 Internet 的一些新特性，使得 VxWorks 的网络性能更加优越，SENS 协议栈层次如图 5-3 所示：

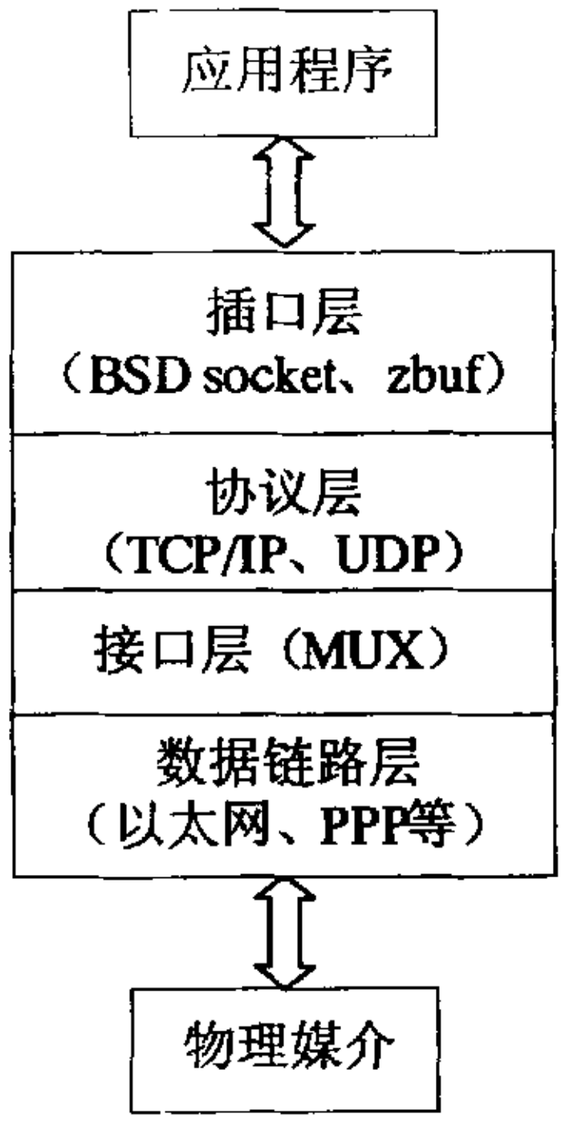


图 5-3　SENS 协议栈[14]

SENS 的基本特征和传统的 TCP/IP 网络协议栈相似，但从图 5-3 中

可以看出 SENS 最大的特点是在数据链路层和网络协议层之间多了 MUX 层。网络接口的驱动程序是叫做 END（Enhanced Network Driver），即增强网络驱动程序，它处于数据链路层。IP 层和 TCP/UDP 层合称为网络协议层。在数据链路层和网络协议层之间有应用程序接口（API），这个接口在 SENS 中叫做 MUX（Multiplexer）接口。MUX 的主要目的是把网络接口驱动和协议层分开，这样就使得网络接口驱动和协议层彼此基本上保持独立。这种独立性使得加载一个新的协议或网络接口驱动变得非常容易。例如：如果要加一个新的网络接口驱动，所有的现有基于 MUX 的协议就都可以用这个新的网络接口驱动程序；同样，如果要加一个新的基于 MUX 的协议，现有的网络接口驱动也能够用 MUX 来与新协议通信。正是利用了 SENS 的这个特性，我们才顺利的通过 MUX 接口在原有的 VxWorks 网络协议栈中添加了 IPv6 协议支持。

SENS 是一个高性能的协议栈，它的优化措施包括取消在 TCP 层的数据拷贝、使用 Hash 表、缓冲管理方法的改进等适合于高性能的网络交换设备到低价的网络接入设备，如 10M/100M 以太网交换机、广域网接入设备、ATM 交换机等。

5.3.2　IPSec 在协议栈中实现结构

IPSec 嵌入 VxWorks 协议栈中与 OS 集成实施时的结构图如下图 5-4 所示，整个 IPSec 实现由密钥管理进程，内核初始化，PF_KEY 接口，数据库管理，数据库，IPSec 输入输出处理和加密验证算法七大功能模块组成（如图 5-4 所示）。

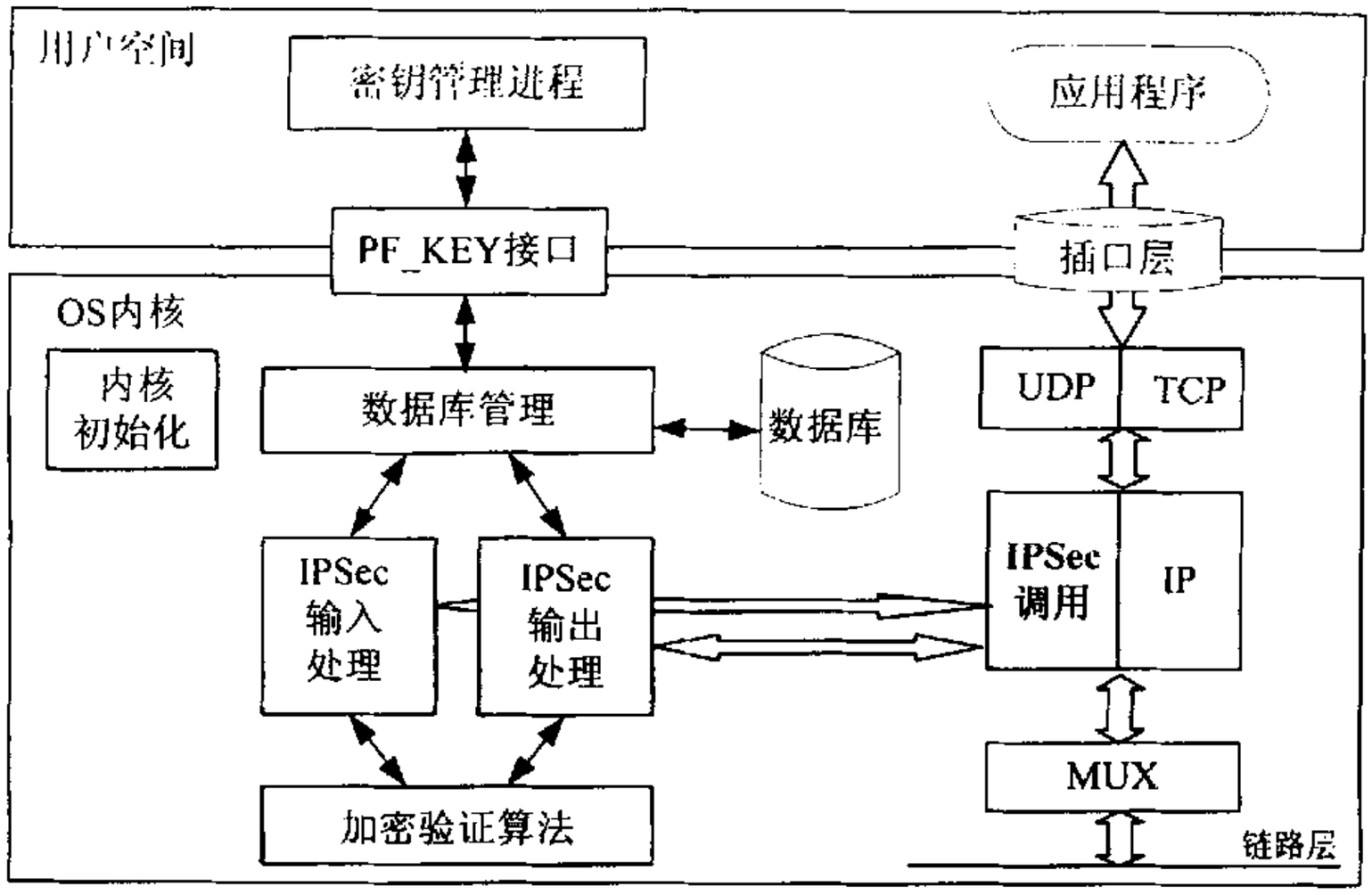


图 5-4　IPSec 在 VxWorks 协议栈中实现结构图

1、密钥管理进程模块：负责手动或者调用 IKE 自动协商生成密钥参数

2、内核初始化模块：向内核注册 IPSec 需要使用的协议

3、PF_KEY 套接字接口模块：用户空间与 OS 内核的接口通道

4、数据库管理模块：对数据库进行添加、删除、查找等操作

5、数据库模块：存放 SA 条目的 SADB 数据库和存放 SP 条目的 SPD 数据库

6、IPSec 输入输出处理模块：对进入和外出数据流进行 IPSec 处理

7、加密验证算法模块：提供加密、验证处理时的算法支持调用

5.3.3　密钥管理进程模块

密钥管理进程模块在用户空间实现，又可以分为手动配置和 IKE 协商两个部分。如图 5-1IPSec 在路由器中实现总体结构图中所示，它与配置子系统相接口，接收用户的配置命令，根据用户的特定安全参数协商创建安全联盟(SA)，为需要进行安全保护的数据包制定安全措施并提供

相应的安全参数。此部分不属于本论文论述的重点，暂不详述。

5.3.4　IPSec 内核初始化模块

5.3.4.1 功能描述

IPSec 内核初始化的目的是把 IPSec 处理时所需要的协议及其对应的协议处理函数在网络协议初始化时注册到 VxWorks 内核协议栈中，以便数据处理时的调用。在 IPSec 的内核处理过程中所涉及到的协议包括 AH、ESP 和一个套接字协议 PF_KEY，我们需要分别对它们进行注册。

5.3.4.2 主要数据结构

SENS 把协议关联到一个域中，并且用一个协议族常量来标识每个域。一个协议域由下面的 domain 结构来表示。

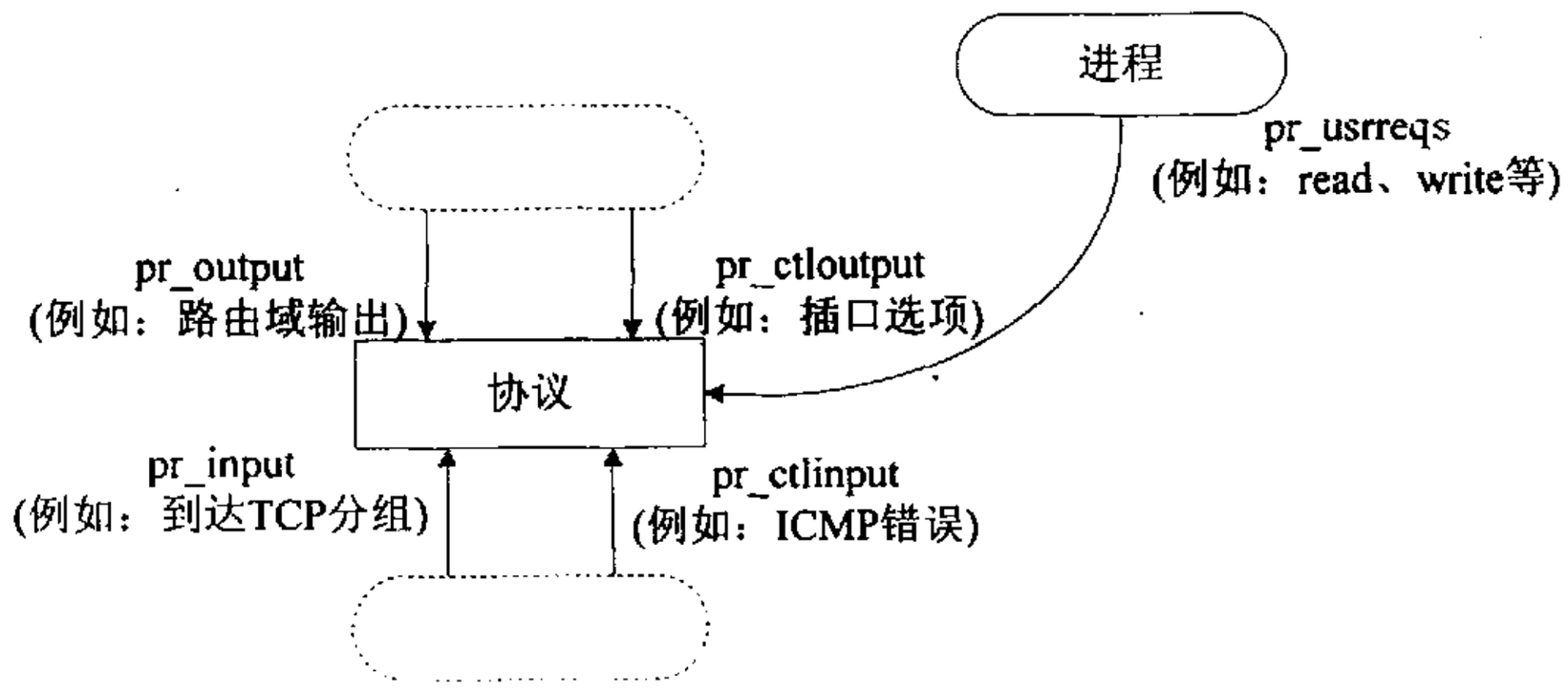
```
struct    domain
{
    int       dom_family;      /* 地址族常量 AF_xxx */
    char      *dom_name;       /*此域的文本名称*/
    int       (*dom_init)();          /* 初始化此域的数据结构 */
    int       (*dom_externalize)();     /*外部接入权限 */
    int       (*dom_dispose)();    /* 内部访问权限*/
    struct    protosw *dom_protosw;
    struct    protosw *dom_protoswNPROTOSW;
    struct    domain *dom_next;
    int       (*dom_rtattach)();    /*此域路由信息*/
    int       dom_rtoffset;
    int       dom_maxrtkey;
};
```

另外，SENS 为内核中的每个协议分配了一个 protosw 结构并初始化，同时将在一个域中的所有协议的这个结构组织到一个数组中。每个 domain 结构引用相应的 protosw 结构数组，一个内核可以通过提供多个 protosw 项为一个协议提供多个接口。protosw 结构定义如下：

```
struct protosw
```

```
{
    short    pr_type;      /* 使用的套接口类型*/
    struct   domain *pr_domain;      /* 相关的域结构*/
    short    pr_protocol;      /*域中协议编号*/
    short    pr_flags;          /* 标识协议的附加特征*/
    /*协议之间的挂钩函数*/
    void     (*pr_input) ();          /*向协议的输入(从底层) */
    int      (*pr_output) ();         /*向协议的输出(从上层) */
    void     (*pr_ctlinput) ();       /*控制输入(从底层) */
    int      (*pr_ctloutput) ();      /*控制输出(从上层) */
    /*用户协议挂钩函数 */
    int      (*pr_usrreq) ();     /*用户请求*/
    void     (*pr_init) ();           /*初始化挂钩函数*/
    void     (*pr_fasttimo) ();       /*快超时(200ms) */
    void     (*pr_slowtimo) ();       /*慢超时 (500ms) */
    void     (*pr_drain) ();
    int      (*pr_sysctl) ();     /*协议系统控制*/
   struct    pr_usrreqs *pr_usrreqs;     /*替代 pr_usrreq() */
};
```

在上述结构体中需要特别注意以下 5 个成员函数指针如图 5-5 所示：



5-5 一个协议的 5 个主要入口点[5]

它们用来提供从其他协议对此协议的访问。pr_input 处理从一个低层协议输入的数据，pr_output 处理从一个高层协议输出的数据，pr_ctlinput 处理来自下层的控制信息，而 pr_ctloutput 处理来自上层的控制信息。

pr_usrreqs 处理来自进程的所有通信请求。

5.3.4.3 协议初始化实现

目前在 VxWorks 内核中已经存在的域有 IP 协议域（inetdomain）和路由协议域（routedomain）。为了使内核支持 IPv6，必须添加 IPv6 协议域（inet6domain），这是基本协议栈部分的工作，在此不再赘述。有了内核双栈支持后，我们只需要把 AH 和 ESP 协议分别注册到 IP 协议域和 IPv6 协议域中即可。但是为了添加 PF_KEY 套接字协议支持，还需要为它重新添加一个域——密钥域（keydomain），再进行密钥协议注册。我们构造的 keydomain 域结构体如下所示：

```
struct domain keydomain =
{ PF_KEY, "key", key_init, 0, 0,(struct protosw *)keysw,
    (struct protosw *)&keysw[sizeof(keysw)/sizeof(keysw[0])],0,0,0,0 };
```

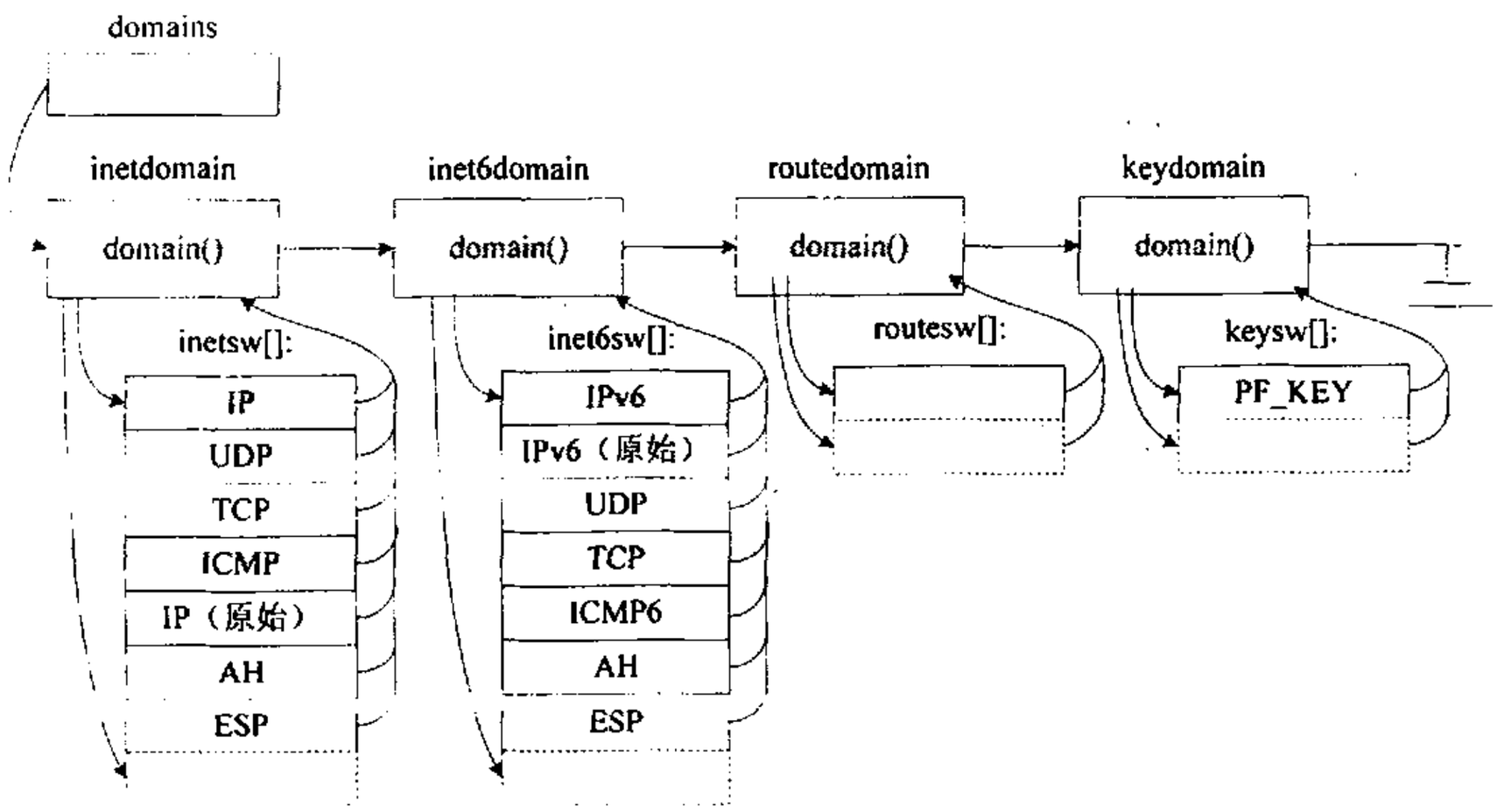


图 5-6 初始化后的 domain 链表和 protosw 数组

在 VxWorks 系统初始化期间，由初始化网络任务的函数 netLibInit()调用 addDomain()来添加各协议域，调用 domaininit()来链接结构 domain

和 protosw，图 5-6 显示了链接的结构 domain 和 protosw，它们配置在内核中来支持 Internet 等协议族。

在 BSD 内核中，所有协议的结构 domain 和 protosw 都是先声明并采用静态初始化的方法，在 VxWorks 中对此有所改进，它采用函数调用的方法动态初始化协议，以便可以根据需要裁减注册的协议，减小内核容量。此类函数处理流程如图 5-7 所示：

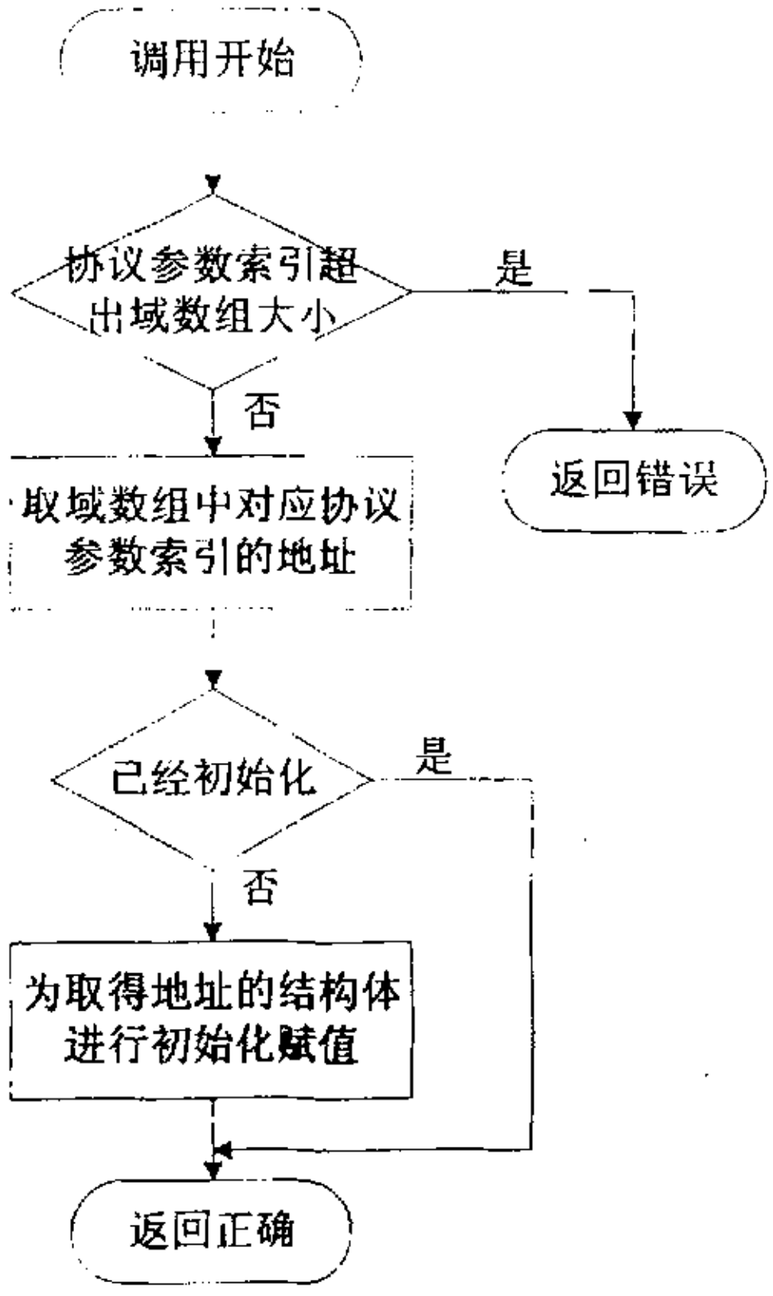


图 5-7 协议动态注册流程图

具体实现函数如下：

STATUS esp4LibInit (void)

功能：向 IP 协议域注册 ESP 协议

STATUS ah4LibInit (void)

功能：向 IP 协议域注册 AH 协议

STATUS esp6LibInit (void)

功能：向 IPv6 协议域注册 ESP 协议

STATUS ah6LibInit (void)

功能：向 IPv6 协议域注册 AH 协议

STATUS keyLibInit (void)

功能：向密钥协议域注册密钥协议

这些注册函数在系统初始化用户任务时根据配置的需要选择调用。

5.3.5　PF_KEY 套接字接口模块

5.3.5.1 功能描述

PF_KEY 套接字是用户空间与内核通信接口，同其他套接字功能相同，它的目的也是将进程发送的与协议相关的请求映射到产生接口时制定的与协议有关的实现，所不同的只是它作为 IKE 与 IPSec 专用的一个密钥套接口。既然作为一个接口，它的实现就应该分为两个部分，在用户空间部分以无差别的系统调用的方式封装起来提供给应用程序，以便其向内核发送消息；而在内核实现的部分就应该区分系统调用针对协议的不同，转化成指定协议的具体实现，负责此次系统调用的真正实施。下图 5-8 说明了进程中的接口与内核中的协议实现之间的层次关系：

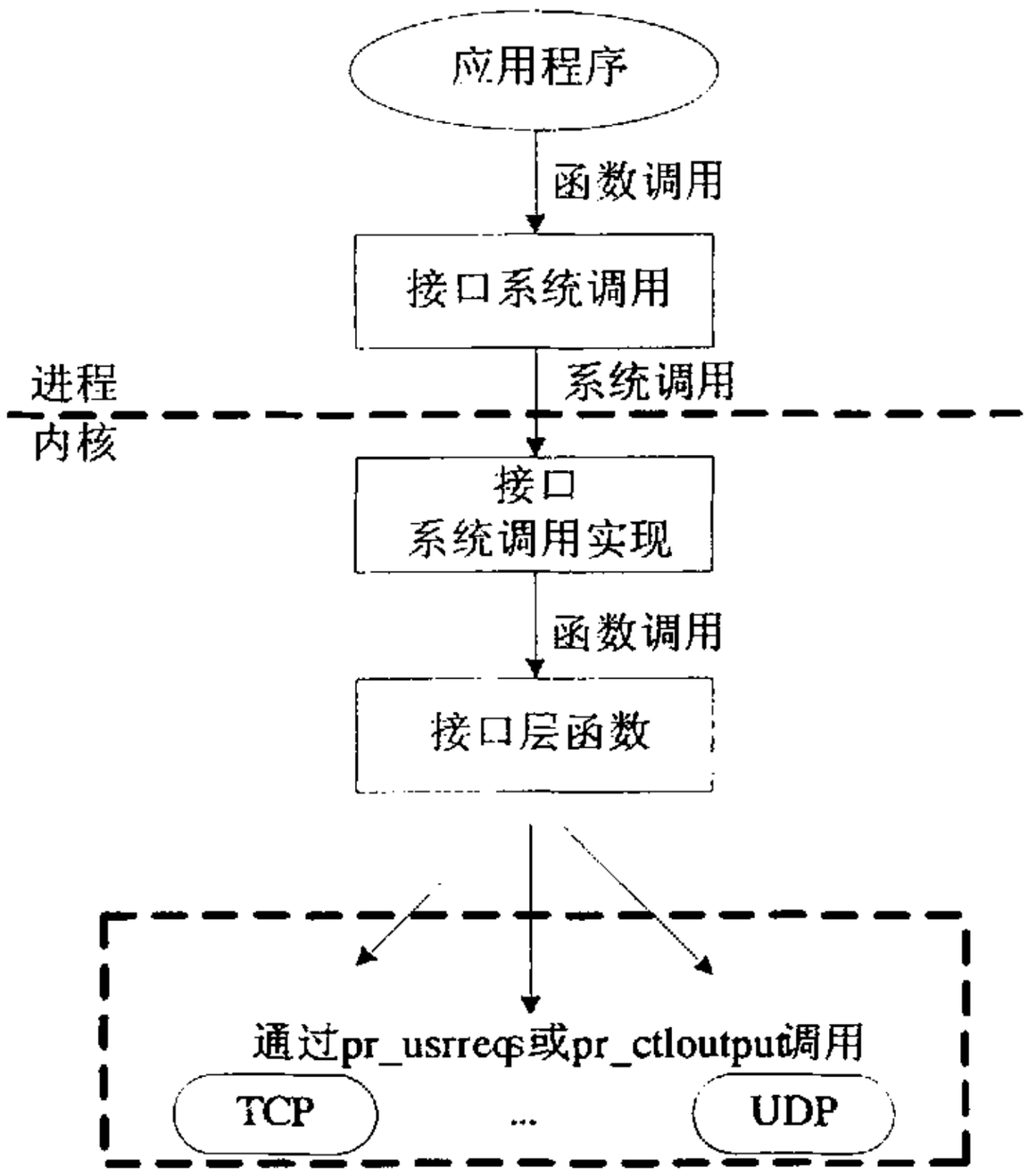


图 5-8 接口层将一般的请求转换为指定的协议操作[5]

我们在此只涉及内核部分的实现，即在内核中如何添加 PF_KEY 套接字支持，以及内核中 PF_KEY 消息处理机制是怎样实现的。

5.3.5.2 主要数据结构

结构体 struct pr_usrreqs 的成员包含了所有网络系统调用的套接字接口层实现函数的指针，如下所示：

```
struct pr_usrreqs {
    int   (*pru_abort) ();
    int   (*pru_accept) ();
    int   (*pru_attach) ();
    int   (*pru_bind) ();
    int   (*pru_connect) ();
    int   (*pru_connect2) ();
    int   (*pru_control) ();
```

```
    int  (*pru_detach) ();
    int  (*pru_disconnect) ();
    int  (*pru_listen) ();
    int  (*pru_peeraddr) ();
    int  (*pru_rcvd) ();
    int  (*pru_rcvoob) ();
    int  (*pru_send) ();
    int  (*pru_sense) ();
    int  (*pru_shutdown) ();
    int  (*pru_sockaddr) ();
    int  (*pru_sosend) ();
    int  (*pru_soreceive) ();
    int  (*pru_sopoll) ();
};

/* PF_KEY 基本消息头结构*/
struct sadb_msg {
   u_int8_t sadb_msg_version;   /* PF_KEY 消息版本号*/
   u_int8_t sadb_msg_type;   /*PF_KEY 消息类型*/
   u_int8_t sadb_msg_errno;   /*PF_KEY 消息错误号*/
   u_int8_t sadb_msg_satype;   /*标识安全联盟类型*/
   u_int16_t sadb_msg_len;   /*消息长度*/
   u_int16_t sadb_msg_reserved;   /*保留字段*/
   u_int32_t sadb_msg_seq;   /*消息的序列号*/
   u_int32_t sadb_msg_pid;   /*消息进程 ID */
};
```

5.3.5.3 PF_KEY 套接口支持实现

在网络应用中支持套接口就是为了使用户程序能够无缝的接入网络服务。为了更容易的添加各种包含套接口支持的网络服务，VxWorks 网络协议组件中引入了一个标准套接字接口。

有了标准套接字接口，就可以添加新的套接字支持层，以便从各自的协议层中接入网络堆栈。这就使得早已熟悉标准套接字 API 的开发者们更加方便的使用各种用户服务。

标准套接字接口可以支持多个不同协议层套接字同时使用，这一点可以由分层结构实现。风河（Wind River）的标准套接字接口层在用户套接字支持层之上，如下图 5-9 所示：

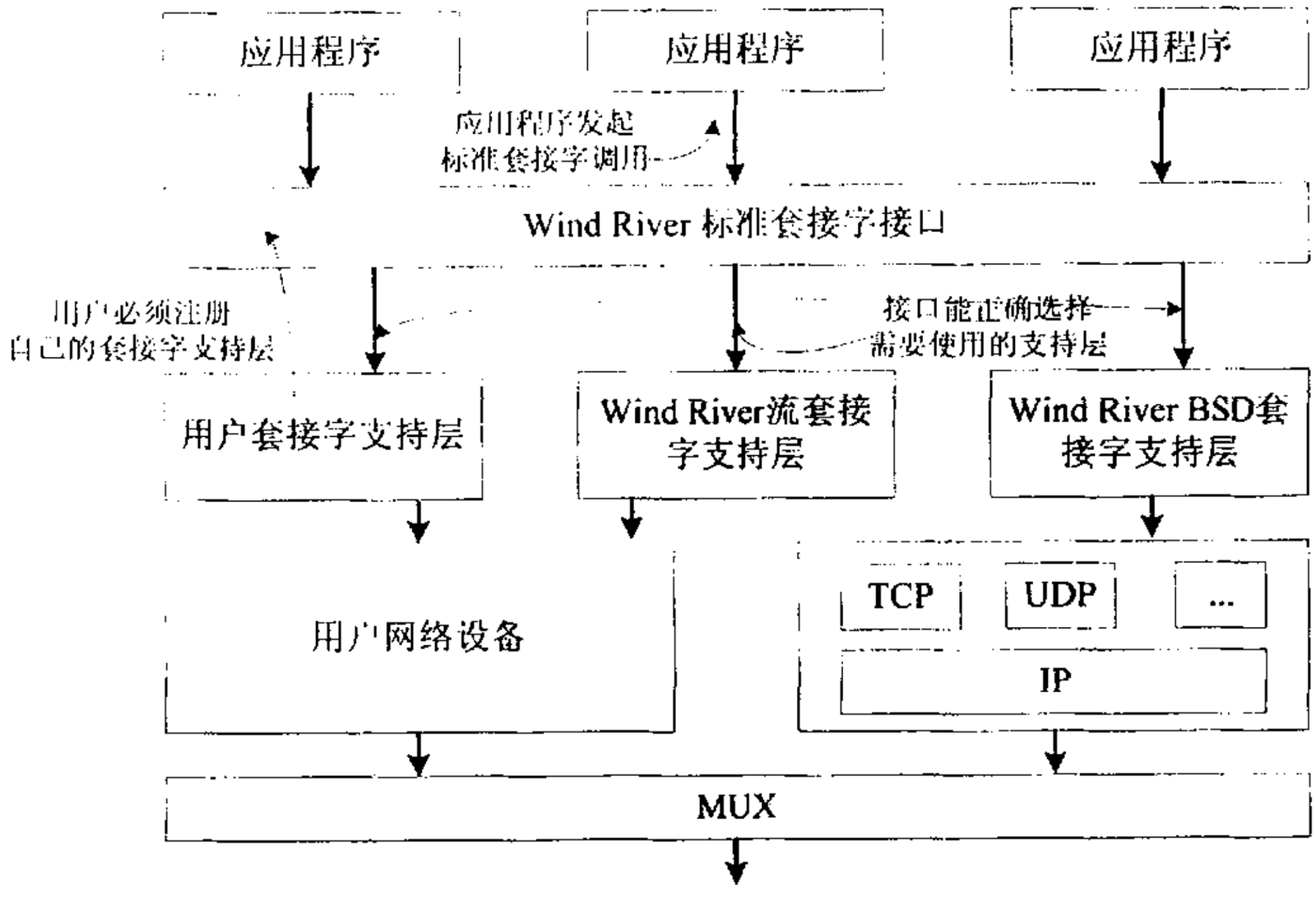


图 5-9 标准套接字接口[18]

为了实现用户套接字支持层，需要用户能够在自己的服务中实施套接字功能并且使系统知道这种新套接字的实施。下面部分就讲述如何做到这一点。

套接字功能接口是一些由特定套接字支持层支持的标准套接字函数的组合。为了创建一个套接字功能接口，可以分两步走，但这两步都必须在网络初始化之前完成。第一步就是创建一个独一无二的用以标识此套接字的常量。我们在 socket.h 头文件中定义 PF_KEY 常量为 27。第二步是添加套接字库，由 sockLibAdd()函数来完成。sockLibAdd()函数被用来在系统套接字执行列表中添加指定域的套接字功能。此函数包含三个

参数。此函数定义如下：

```
STATUS sockLibAdd
(
    FUNCPTR  sockLibInitRtn,      /* back-end init routine */
    int      domainMap,     /* address family */
    int      domainReal     /* address family */
)
```

SockLibInitRtn 参数指向调用 sockLibAdd()函数时启动的套接字库初始化函数，此函数会向系统注册套接字功能函数表 SOCK_FUNC。SOCK_FUNC 表包含了 19 个常用的 sockets 函数指针。

DomainMap 参数是一个用来独一无二的标识此套接字执行的协议族标识符。

DomainReal 是为此套接字的实施独一无二的定义地址域的常量。

我们在初始化 BSD 套接口函数中调用 sockLibAdd()来实现对 PF_KEY 套接字的添加。

新的用户套接字被添加完成后，在具体处理过程中，应用程序的套接字调用将根据创建套接字时传送给 socket()函数调用的 domain 参数被定向到正确的底层套接字支持点。如果这个参数匹配调用 sockLibAdd()程序添加新套接字支持点时使用的 domainMap 参数，那么此套接字调用被定向到这个新的支持点。因为用户注册自己的套接字时，用户向系统提交了 SOCK_FUNC 表，这张表格保存了用户为了支持自己的应用而创建的可以被调用的套接字函数。于是系统就能够支持使用用户注册的标准套接字调用。

为了实现 PF_KEY 套接字，我们定义了自己的 SOCK_FUNC 表——key_usrreqs，其结构如下所示。此表中保存了 PF_KEY 的套接口函数指

针。因为目前 PF_KEY 域支持的是 SOCK_RAW 类型，它的套接口基本处理函数也就都是从原始套接口的处理衍生而来。

```
struct pr_usrreqs key_usrreqs = {
    key_abort, pru_accept_notsupp, key_attach, key_bind,key_connect,
    pru_connect2_notsupp, pru_control_notsupp, key_detach,
    key_disconnect, pru_listen_notsupp, key_peeraddr,pru_rcvd_notsupp,
    pru_rcvoob_notsupp, key_send, pru_sense_null, key_shutdown,
    key_sockaddr, sosend, soreceive, 0
};
```

为了向系统提交此 SOCK_FUNC 表，我们在系统初始化过程中向 keydomain 域注册 PF_KEY 协议时，将 key_usrreqs 结构体指针传递给了密钥域的协议数组 keysw[]，它是一个 protosw 结构体类型的指针数组，key_usrreqs 指针就是赋给 protosw 结构的 pr_usrreqs 成员，此成员项是用于处理来自进程对协议的所有通信请求的。

5.3.5.4 PF_KEY 内核消息机制实现

用户进程通过使用这个 socket 接口来发送和接收信息与内核通信。为了传递消息，PF_KEY 定义了基本消息头和扩展消息头，并制定了一套自己的消息通信机制，这在前面已经有了介绍。PF_KEY 的消息由基本头和扩展数据构成，基本头后紧跟扩展数据，扩展数据由一个或多个扩展头组成。正常情况下，正确格式的消息发给内核，内核将应答消息返回到 PF_KEY socket 中。如果内核检测到错误，错误标识会随应答消息一起被发送出去。下面讲述内核接收和发送消息的具体处理过程。

1、内核接收消息处理函数

如上所述，protosw 结构体的 pr_output 成员是用于注册处理从一个高层协议输出数据的函数指针，我们将内核接收应用层消息的处理函数

key_output 在协议注册时赋值给 pr_output 成员项，这样，当密钥管理进程从用户空间发送消息给内核时，就交由 key_output 函数来处理。key_output 函数的具体处理流程如下所图 5-10 所示：

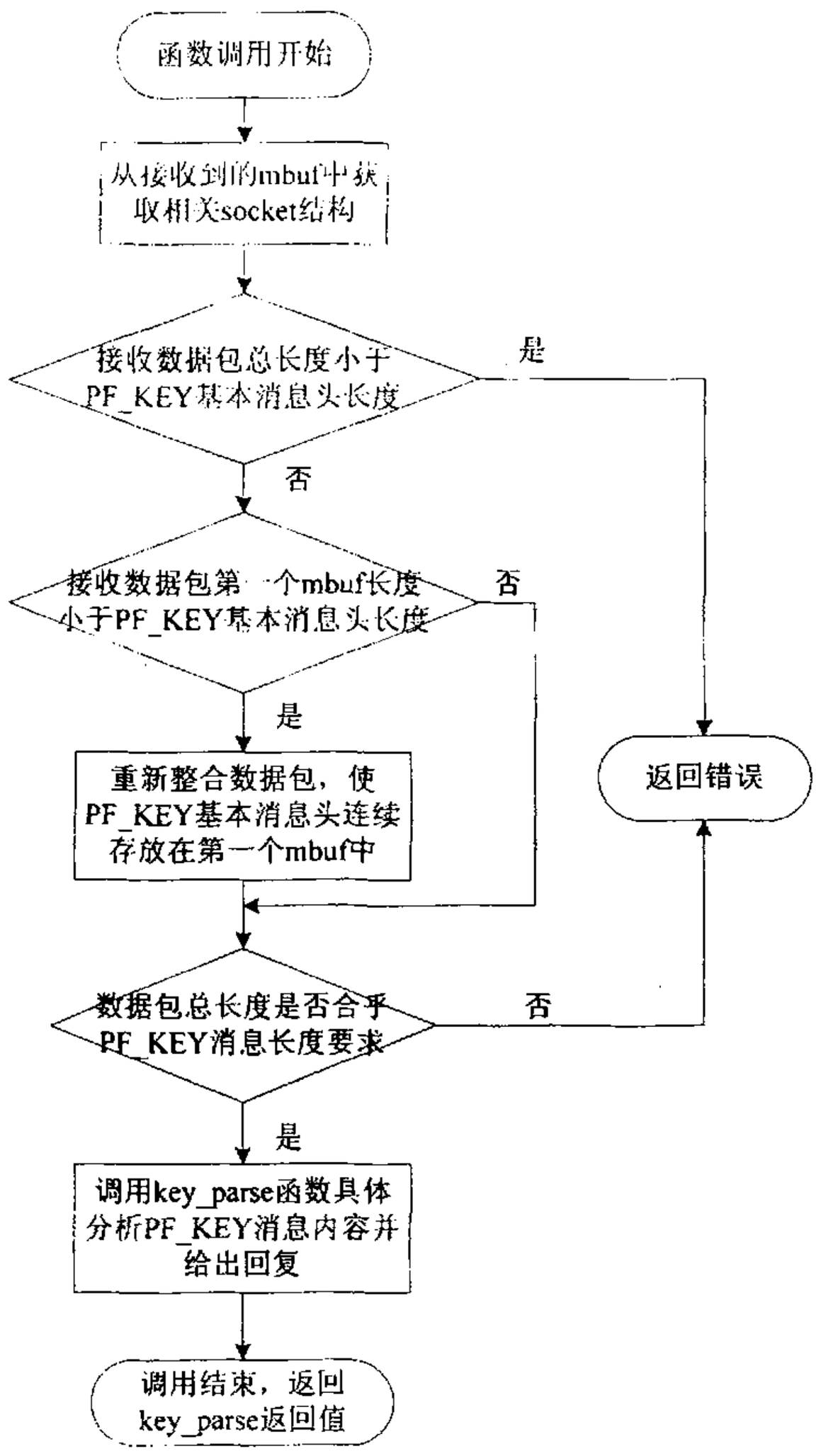


图 5-10　内核接收消息处理流程图

key_output 函数的主要任务就是对从上层接收到的数据包进行合法性分析，是否符合正确的 PF_KEY 消息要求，并且必要时对数据包进行重新整合，便于以后进一步的分析处理。而对接收到的消息的真正分析处理就交由下层函数 key_parse 去实现。

2、内核发送消息处理函数

内核对来自密钥管理进程的 PF_KEY 消息处理完毕后，需要对其进行消息回复，如果发生错误，也要向用户空间返回错误信息。内核向应用程序发送消息的处理工作由 key_sendup_mbuf 函数来完成。此函数的具体工作流程如下图 5-11 所示：

key_sendup_mbuf 函数的主要任务就是对发送的消息进行合法性检查，必要时整合数据包，为需要进行发送的数据包寻找正确的协议控制块以便将其发送到正确的套接字接口上。

5.3.6　数据库管理模块

5.3.6.1 功能描述

数据库管理功能模块主要面向密钥管理模块的各种数据库操作请求，它处理这些请求，操纵数据库，组织填写信息，进行 SADB 和 SPD 的维护和查询等操作。

由于 PF_KEY 协议为在应用层运行的密钥管理程序和其它管理 SADB、SPD 的工具提供了一种操作在内核中建立与 SADB 和 SPD 的通信，因此，对数据库的管理的各种维护操作如添加、删除、修改等都是建立在 PF_KEY 消息基础上的，也就是主要针对来自用户空间的 PF_KEY 消息进行分析，再根据各种消息类型按需求调用各种相应的处理，必要时给出应答。

对数据库的查询操作是在数据包的输入输出处理过程中进行的。

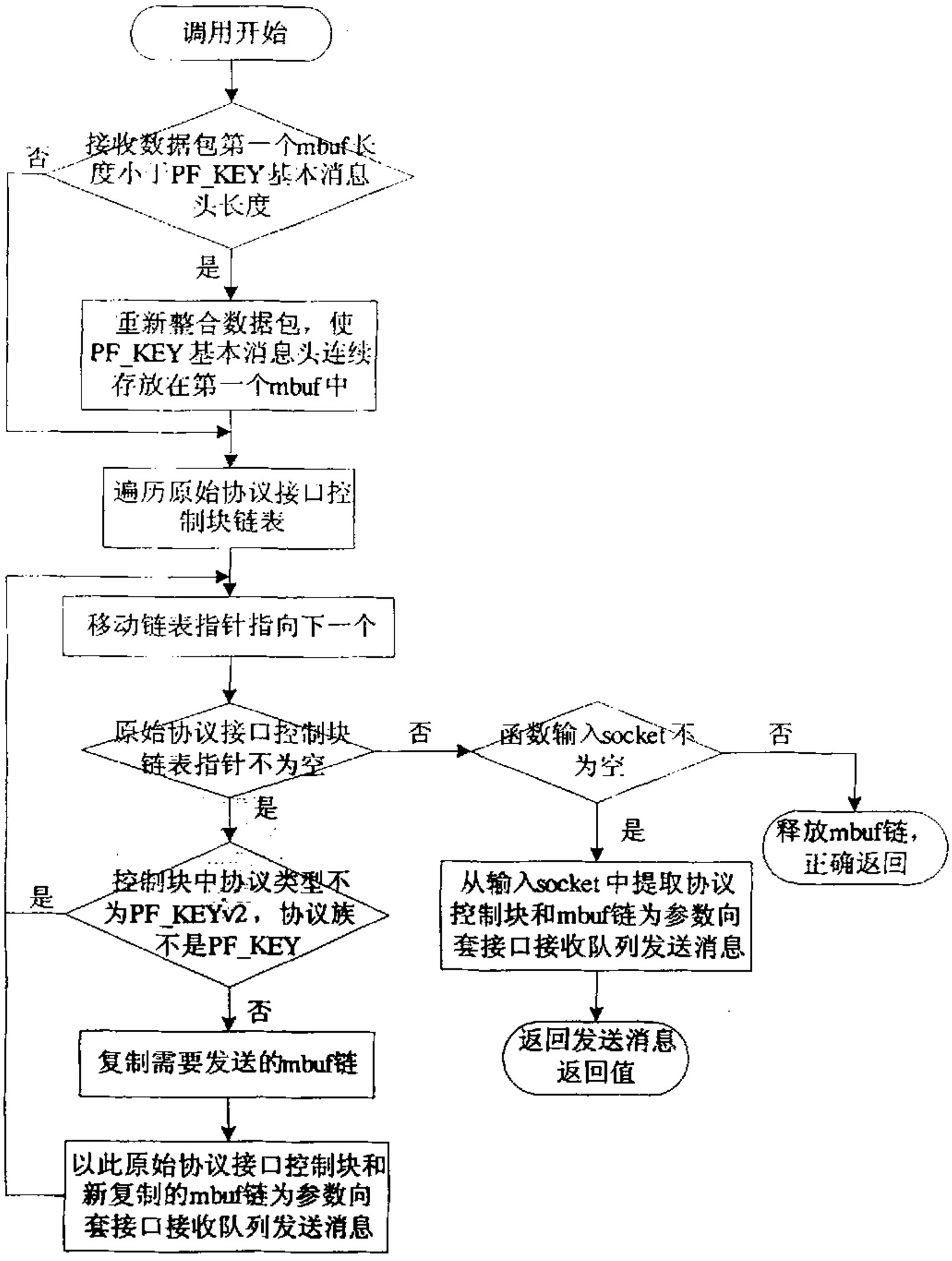


图 5-11 内核发送消息处理流程图

5.3.6.2 主要数据结构

静态函数指针结构体数组 key_typesw 中保存了针对各种 PF_KEY 消

息类型的处理函数指针，如下所示。

```
static int (*key_typesw[]) (struct socket *, struct mbuf *,
        const struct sadb_msghdr *) = {
    NULL,       /* SADB_RESERVED */
    key_getspi,  /* SADB_GETSPI */
    key_update, /* SADB_UPDATE */
    key_add,    /* SADB_ADD */
    key_delete,  /* SADB_DELETE */
    key_get,/* SADB_GET */
    key_acquire2,    /* SADB_ACQUIRE */
    key_register,/* SADB_REGISTER */
    NULL,       /* SADB_EXPIRE */
    key_flush,   /* SADB_FLUSH */
    key_dump,   /* SADB_DUMP */
    key_promisc,    /* SADB_X_PROMISC */
    NULL,       /* SADB_X_PCHANGE */
    key_spdadd, /* SADB_X_SPDUPDATE */
    key_spdadd, /* SADB_X_SPDADD */
    key_spddelete,   /* SADB_X_SPDDELETE */
    key_spdget,  /* SADB_X_SPDGET */
    NULL,       /* SADB_X_SPDACQUIRE */
    key_spddump,    /* SADB_X_SPDDUMP */
    key_spdflush,    /* SADB_X_SPDFLUSH */
    key_spdadd, /* SADB_X_SPDSETIDX */
    NULL,       /* SADB_X_SPDEXPIRE */
    key_spddelete2,  /* SADB_X_SPDDELETE2 */
};
```

5.3.6.3 数据库管理实现

上一节提到内核接收到 PF_KEY 消息后，没有进行具体的消息处理工作，而是调用 key_parse 函数继续以后的处理，key_parse 函数就是数据库管理操作的总入口处，它的处理流程图如下图 5-12 所示：

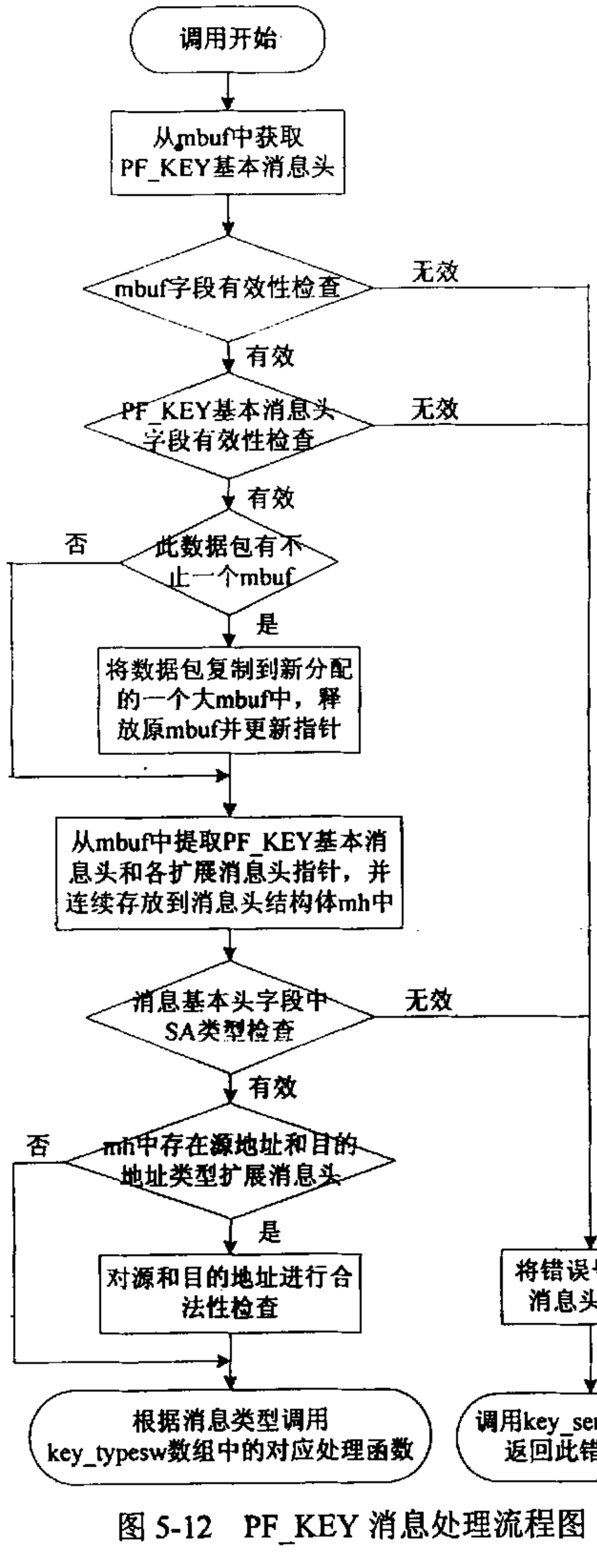


图 5-12　PF_KEY 消息处理流程图

可见，key_parse 的工作就是对用户空间的 PF_KEY 消息进行进一步的处理，负责消息分类，调用各自的实施函数，发生错误时回复错误信息。这些函数的调用关系可总结如下图 5-13 中所示：

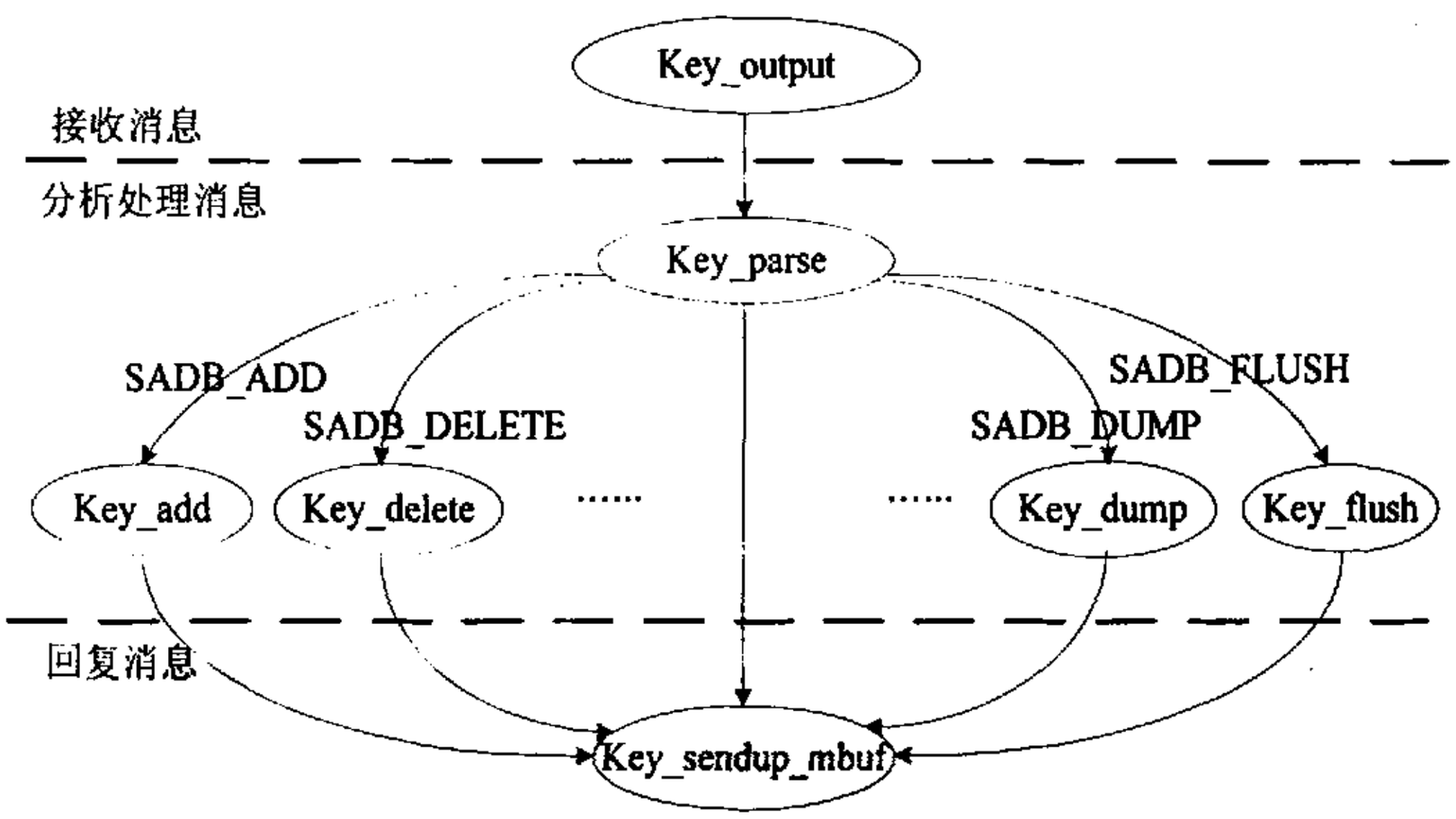


图 5-13　PF_KEY 消息处理函数调用关系图

5.3.7　数据库模块

5.3.7.1 功能描述

数据库模块中包含了安全策略数据库（SPD）和安全联盟数据库（SADB）。

SPD 说明了对 IP 数据报提供何种保护，并以何种方式实施保护。SPD 中策略项的建立和维护应通过协商；而且对于进入和外出处理都应该有自己的策略库。对于进入或外出的每一份数据报，都可能有三种处理：丢弃、绕过或应用 IPSec。SPD 提供了便于用户或系统管理员进行维护的管理接口。可允许主机中的应用程序选择 IPSec 安全处理。SPD 中的策略项记录对 SA(SA 束)进行了规定，其字段包含了 IPSec 协议、模式、算法和嵌套等要求。SPD 还控制密钥管理(如 ISAKMP)的数据包，即对

ISAKMP 数据包的处理明确说明。

SADB 维护了 IPSec 协议用来保障数据保安全的 SA 记录。每个 SA 都在 SADB 中有一条记录相对应。对于外出处理，应在 SPD 中查找指向 SADB 中 SA 的指针，如 SA 未建立，则应激活 IKE 建立 SA，并同 SPD 和 SADB 的记录关联起来。对于进入处理，SAD 的记录用目的 IP 地址、IPSec 协议类型和 SPI 标识，进行详细匹配找到相应的 SA。SA 的管理可以通过手工进行，也可以通过 IKE 来进行动态协商。

SPD 和 SADB 的结构实现借助了系统内核提供的一套 LIST_*链表处理机制。它自己定义了一种链表头存储结构 LIST_HEAD(name,type)和链表条目存储结构 LIST_ENTRY(type)；并提供了一整套的链表操作宏定义函数，包括对链表的添加（LIST_INSERT_*）、删除（LIST_REMOVE）、查询（LIST_FOREACH）等。其具体定义见<sys/Queue.h>文件。

5.3.7.2 主要数据结构

```
/*链表头指针结构*/
#define LIST_HEAD(name, type)                \
struct name {                \
    struct type *lh_first;  /*第一个元素*/            \
}

/*双向链表结构*/
#define LIST_ENTRY(type)        \
struct {        \
    struct type *le_next;  /*下一个元素*/        \
    struct type **le_prev;/*前一个元素指针地址*/     \
}

/*安全策略索引*/
struct secpolicyindex {
    u_int8_t dir;              /*数据包流动的方向*/
    struct sockaddr_storage src;    /*安全策略的源地址*/
```

```
    struct sockaddr_storage dst;   /*安全策略的目的地址  */
    u_int8_t prefs;              /*源地址的前缀长度*/
    u_int8_t prefd;              /*目的地址的前缀长度*/
    u_int16_t ul_proto;          /*上层协议*/
};

/*安全策略数据库*/
struct secpolicy {
    LIST_ENTRY(secpolicy) chain;
    int refcnt;                  /*引用计数*/
    struct secpolicyindex spidx;   /*选择符*/
    u_int32_t id;                /*独一无二的 ID 号*/
    u_int state;                 /* 0: 死亡, 其它: 存活  */
    u_int policy;        /* DISCARD, NONE  或 IPSEC*/
    struct ipsecrequest *req; /*  对应安全请求条目指针*/
    /*生存时间处理参数*/
    long created;        /*创建策略的时间*/
    long lastused;           /*  最后使用时间，随内核发包更新*/
    long lifetime;           /*  此策略的生存时间  */
    long validtime;          /*  在不再被使用情况下的生存时间*/
};

/*安全联盟索引*/
struct secasindex {
    struct sockaddr_storage src;   /*SA 的源地址  */
    struct sockaddr_storage dst;   /* SA 的目的地址*/
    u_int16_t proto;         /* IPPROTO_ESP  或 IPPROTO_AH */
    u_int8_t mode;               /*  协议模式*/
    u_int32_t reqid;         /*SA 所属者 ID*/
};

/*  安全联盟数据库*/
struct secashead {
    LIST_ENTRY(secashead) chain;
    struct secasindex saidx;
    struct sadb_ident *idents; /*  源身份*/
    struct sadb_ident *identd; /*  目的身份*/
```

```
    u_int8_t state;              /* MATURE 或 DEAD. */
    LIST_HEAD(_satree, secasvar) savtree[SADB_SASTATE_MAX+1];
                                 /* SA 链*/
    struct route sa_route;       /* 路由缓存*/
};

/* 安全联盟*/
struct secasvar {
    LIST_ENTRY(secasvar) chain;
    int refcnt;                  /* 引用计数*/
    u_int8_t state;              /* 安全联盟的状态*/
    u_int8_t alg_auth;           /*验证算法标识符*/
    u_int8_t alg_enc;            /* 加密算法标识符*/
    u_int32_t spi;               /* SPI 值，网络字节序*/
    u_int32_t flags;             /* 为 SADB_KEY_FLAGS 保留*/
    struct sadb_key *key_auth;   /*验证密钥*/
    struct sadb_key *key_enc;    /*加密密钥*/
    caddr_t iv;                  /* 初始化向量 IV*/
    u_int ivlen;                 /* IV 长度*/
    void *sched;                 /*中间加密密钥*/
    size_t schedlen;
    struct secreplay *replay;    /*抗重播保护*/
    long created;
    struct sadb_lifetime *lft_c;  /* 当前生存时间 */
    struct sadb_lifetime *lft_h;  /* 硬生存时间*/
    struct sadb_lifetime *lft_s;  /* 软生存时间*/
    u_int32_t seq;               /*序列号*/
    pid_t pid;                   /*消息 ID*/
    struct secashead *sah;       /* 回指安全联盟数据库条目 */
};
```

5.3.7.3 SPD 结构实现

在 SPD 结构实现中，我们定义了一个三元静态指针数组 sptree，如下所示：static LIST_HEAD(_sptree, secpolicy) sptree[IPSEC_DIR_MAX]。它的每个元素是一个 SPD 链表头指针。我们把 SP 条目项按照其处理方向分类，处理方向的定义如下表 5-1 所示：

方向类型	宏定义	宏定义值
任意方向	IPSEC_DIR_ANY	0
进入方向	IPSEC_DIR_INBOUND	1
外出方向	IPSEC_DIR_OUTBOUND	2

表 5-1 SPD 中数据包处理方向定义

每个方向的 SP 条目链在一起组成一个双向链表，其链表头指针就赋值给 sptree 中以对应方向为下标的元素。其链表结构如下图 5-14 所示：

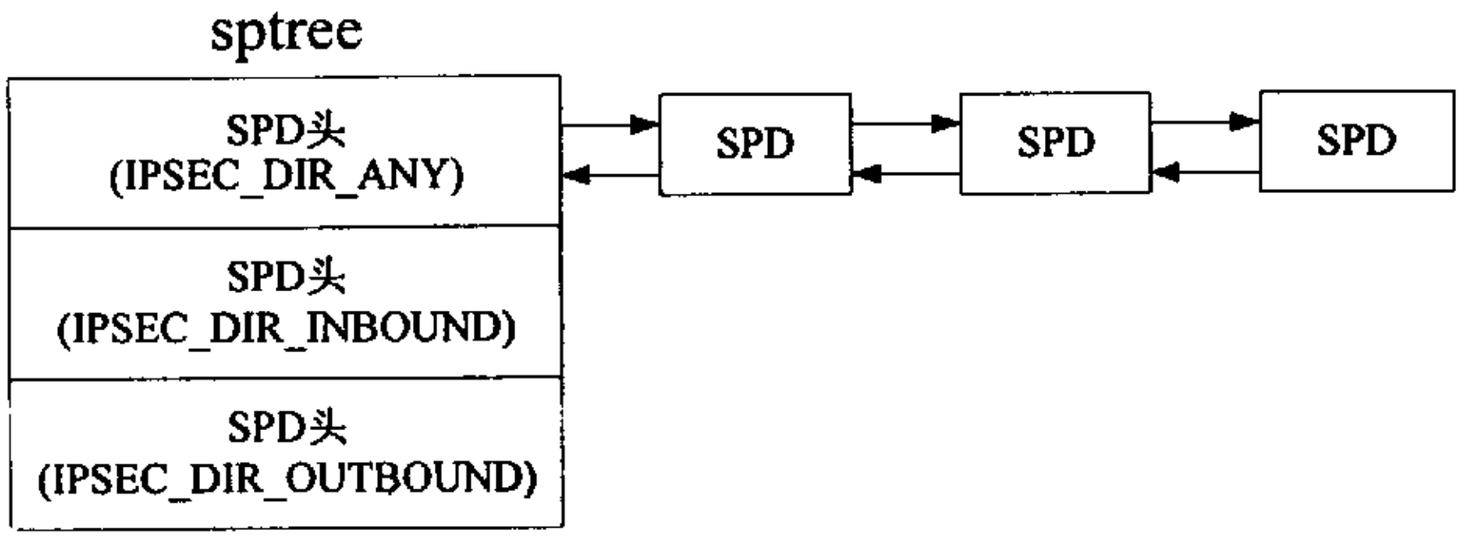


图 5-14　SPD 结构图

在链表中，对SPD条目的操作首先要根据SPD条目中的方向在sptree中定位它所要在的链表头指针，然后在此方向的双向链表中根据它的安全策略索引 spidx（包括源地址、目的地址等）或 ID 值确定其是否已经存在，如果需要添加新条目，就直接返回，否则，将新的 SPD 条目加入到链表的末尾；如果需要删除此条目，就可以执行删除操作；如果需要查询，返回此条目所在指针位置。

5.3.7.4 SADB 结构实现

我们在组织 SADB 的结构时，也定义了一个保存 SADB 链表头指针的一元静态结构体数组 sahtree，如下所示：static LIST_HEAD(_sahtree, secashead) sahtree;SADB 条目以双向链表的形式连接起来，为了便于查

找，其链表头指针就赋值给 sahtree。而 SADB 条目中也存在一个四元静态指针数组 savtree，如下所示：LIST_HEAD(_satree, secasvar) savtree[SADB_SASTATE_MAX+1]; 它的每个元素是一个 SA 条目链表头指针。我们把 SA 条目项按照其 SA 状态类型分类，其 SA 状态的定义如下表 5-2 所示：

状态类型	宏定义	宏定义值
未成熟状态	SADB_SASTATE_LARVAL	0
成熟状态	SADB_SASTATE_MATURE	1
垂死状态	SADB_SASTATE_DYING	2
死亡状态	SADB_SASTATE_DEAD	3

表 5-2　SADB 中 SA 状态定义

每种状态的 SA 条目链在一起组成一条双向链表，其链表头指针就赋值给 savtree 中以对应方向为下标的元素。其链表结构如图 5-15 所示：

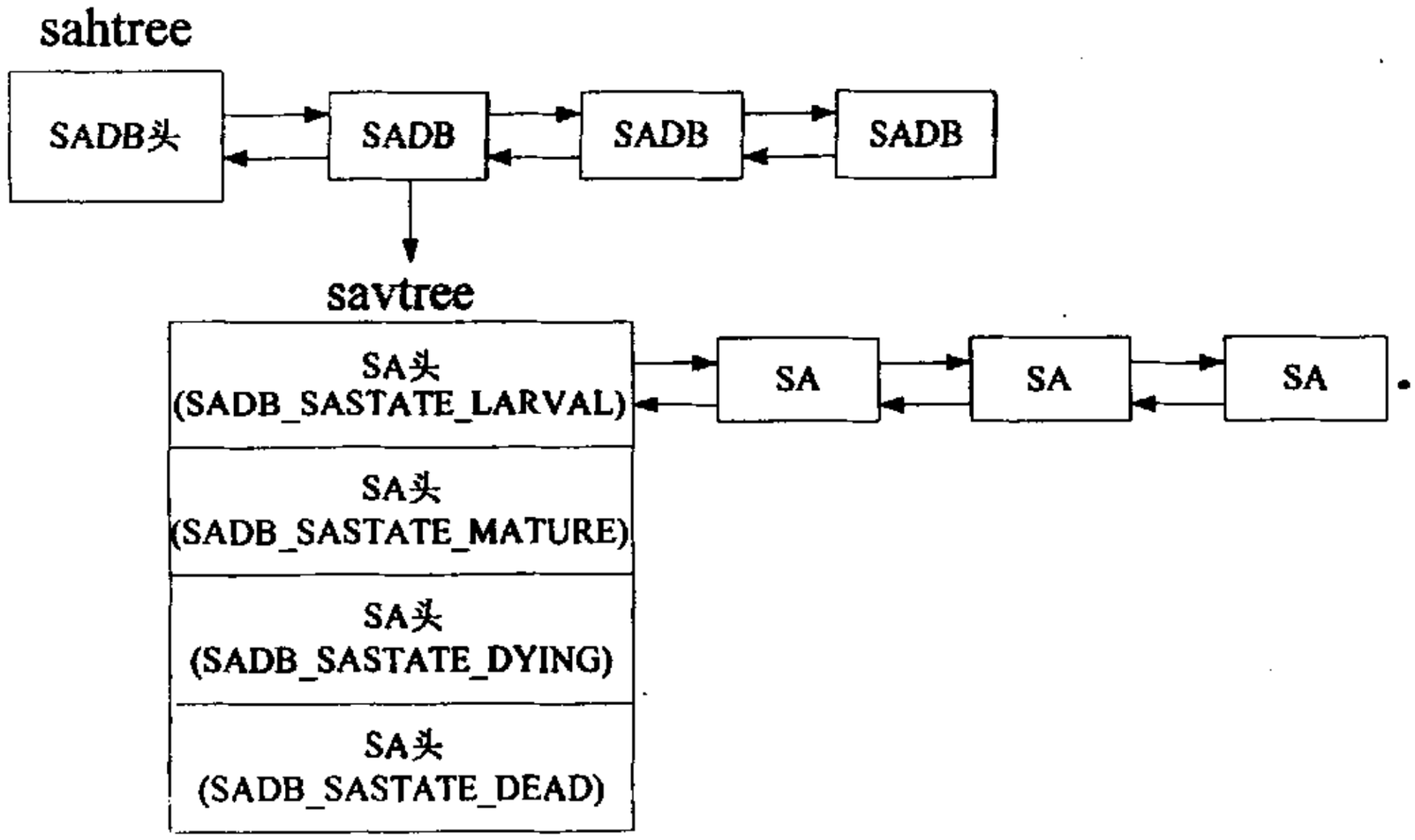


图 5-15　SADB 结构图

如此组织 SADB 的结构主要是根据实际情况的需要，一条安全策略

中可能需要几条 SA，即组成 SA 集束来实现安全功能，而将 SA 条目链表按照 SA 所处状态类型来分类，可以提高查找速度。在链表中，对 SADB 条目的操作就是一些双向链表的操作：以安全参数索引 saidx（包括源地址、目的地址、协议等）为关键字顺序查找，表头添加。而对 SA 条目的操作则先要根据 SA 条目中的 SA 状态类型在 savtree 中定位它所要在的链表头指针，然后在此类型的双向链表中根据它的 SPI 值确定其是否已经存在，如果需要添加新条目，就直接返回，否则，将新的 SA 条目加入到链表的首部；如果需要删除此条目，就可以执行删除操作；如果需要查询，返回此条目所在指针位置。

5.3.8　IPSec 输入输出处理模块

5.3.8.1 功能描述

IPSec 输入输出处理模块是 IPSec 处理的主体模块，负责对流经 IP 层数据包的安全处理：对从数据链路层输入的 IPSec 数据包进行验证、解密等解封装处理；对从传输层输出的数据包进行 IPSec 验证、加密等封装处理。

具体来说，当从数据链路层接收到数据包时，先判断收到的包中有无包含 IPSec 头，如果没有，就将包传递给下一层，作进一步的处理。如果 IP 包中包含了 IPSec 头，就会调用 IPSec 输入处理例程对这个包进行处理。此 IPSec 头的协议值要么是 AH ，要么是 ESP 。根据这个协议值，选择此数据包是由 AH，还是由 ESP 进行处理。不论采用哪个协议处理，它都会从 IP 数据报中提取出 SPI 、源地址和目标地址，然后利用<SPI, 目标地址, 协议>字符组对 SADB 数据库进行检索。根据所得的 SA 条目参数，进行验证处理或者是解密处理。协议载荷处理完之后，还需要查询策略，对载荷进行校验。

对于外出的数据包，首先要查询 SPD，确定为数据包应使用的安全策略，有三种可能：丢弃—丢弃数据包，记录出错信息；绕过—给数据包添加 IP 头，然后发送；应用—查询 SADB，看是否存在有效 SA：如果存在有效的 SA，则取出相应的参数，将数据包封装（包括加密、验证，添加 IPSec 头和 IP 头等），然后发送。若尚未建立 SA，通过数据库管理模块调用 IKE 进行密钥协商，此数据包就会被放到等待队列中，等到协商成功 SA 建好之后，就会增添适当的 AH 或 ESP 头，对包进行封装发送处理，不成功则应将数据包丢弃，并记录出错信息。

5.3.8.2 主要数据结构

```
/*新的 AH 包头*/
struct newah {
    u_int8_t ah_nxt;        /*下一个头字段*/
    u_int8_t ah_len;        /*AH 头长度*/
    u_int16_t    ah_reserve;  /*  保留值*/
    u_int32_t    ah_spi;        /*安全参数索引*/
    u_int32_t    ah_seq;        /*  序列号*/
    /* variable size, 32bit bound*/      /*  验证数据，可变长*/
};

/*新的 ESP 包头*/
struct newesp {
    u_int32_t    esp_spi; /*  安全参数索引*/
    u_int32_t    esp_seq;/*序列号*/
    /*variable size*/       /* (IV 和) 载荷数据*/
    /*variable size*/       /*  填充*/
    /*8bit*/            /*  填充大小*/
    /*8bit*/            /*  下一个头*/
    /*8bit*/
    /*variable size, 32bit bound*/      /*  验证数据，可变长*/
};

/*ESP 尾*/
```

```
struct esptail {
    u_int8_t esp_padlen; /*填充长度 */
    u_int8_t esp_nxt;/*下一个头*/
    /*variable size, 32bit bound*/ /* 验证数据*/
};
```

5.3.8.3 IP 模块输入输出的实现

在 IP 层接收包的过程中，以太网设备驱动程序将把它们接收到的帧传递给 ether_input 做进一步处理，而 ether_input 会判断接收到的数据的类型，并将接收到的分组加入到网络任务队列中等待处理。如下图 5-16 所示：

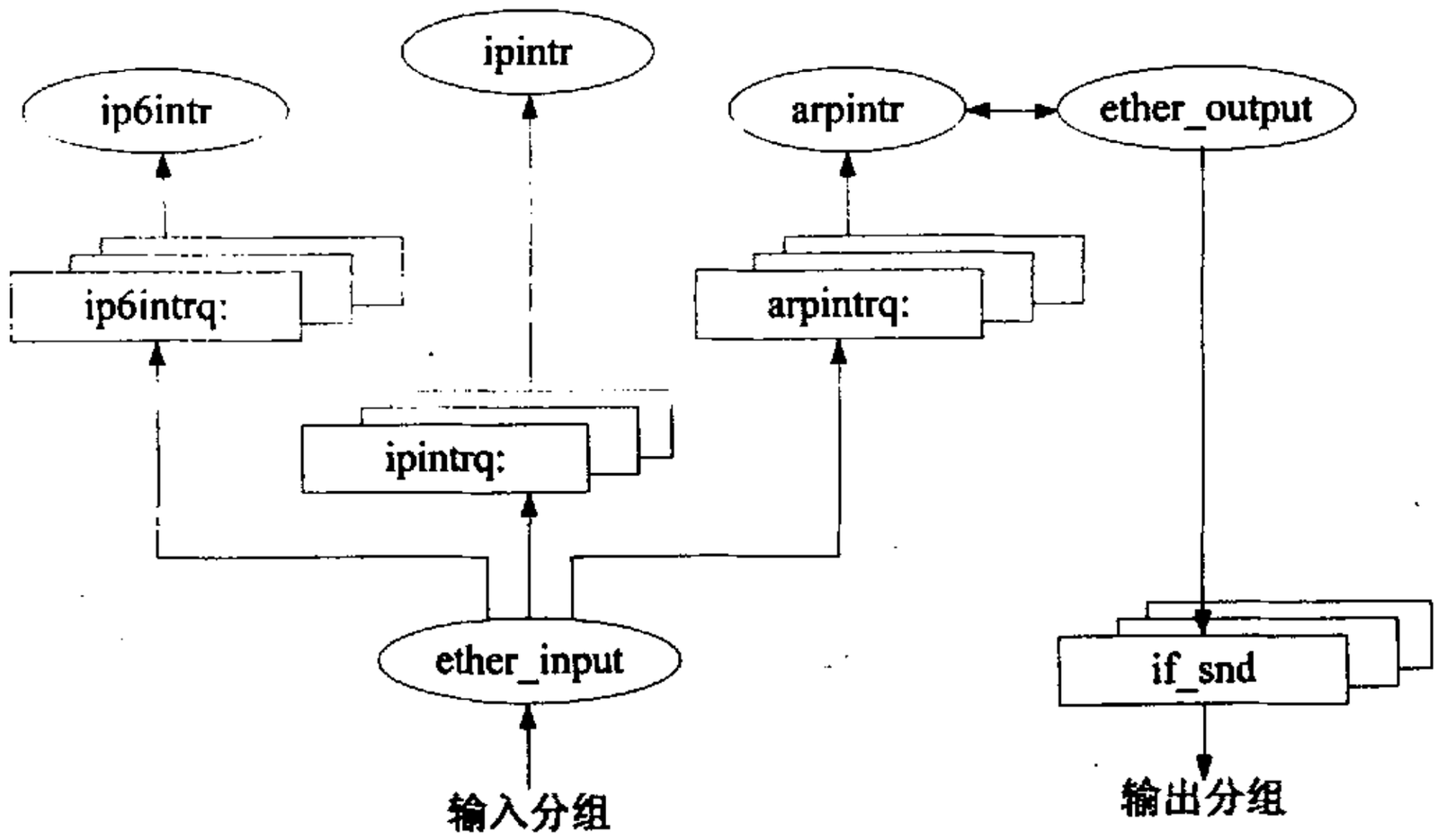


图 5-16　IP 网络队列收发包示意图[5]

当接口把分组放到队列上排队后，通过 schednetisr 调用一个软中断。当该软中断发生时，通过判断 schednetisr 调度的具体队列，如果 IP 处理过程已经由 schednetisr 调度，则内核调用 IP 输入例程 ipintr 处理 IP 数据包。依此类推，调用 ip6intr 来处理 IPv6 输入对列中的分组，调用 arpintr 来处理 ARP 输入队列中的分组。当一个网络层协议，如 IP，调用此接口开始输出时，由 ether_output 来构造帧，并将这个帧放置到接口到发

送队列中，等待以太网卡的发送。

下面我们分协议介绍基本的 IP 处理过程，包括输入、转发和输出。

IPv4 协议栈的处理流程如下图 5-17 中所示：

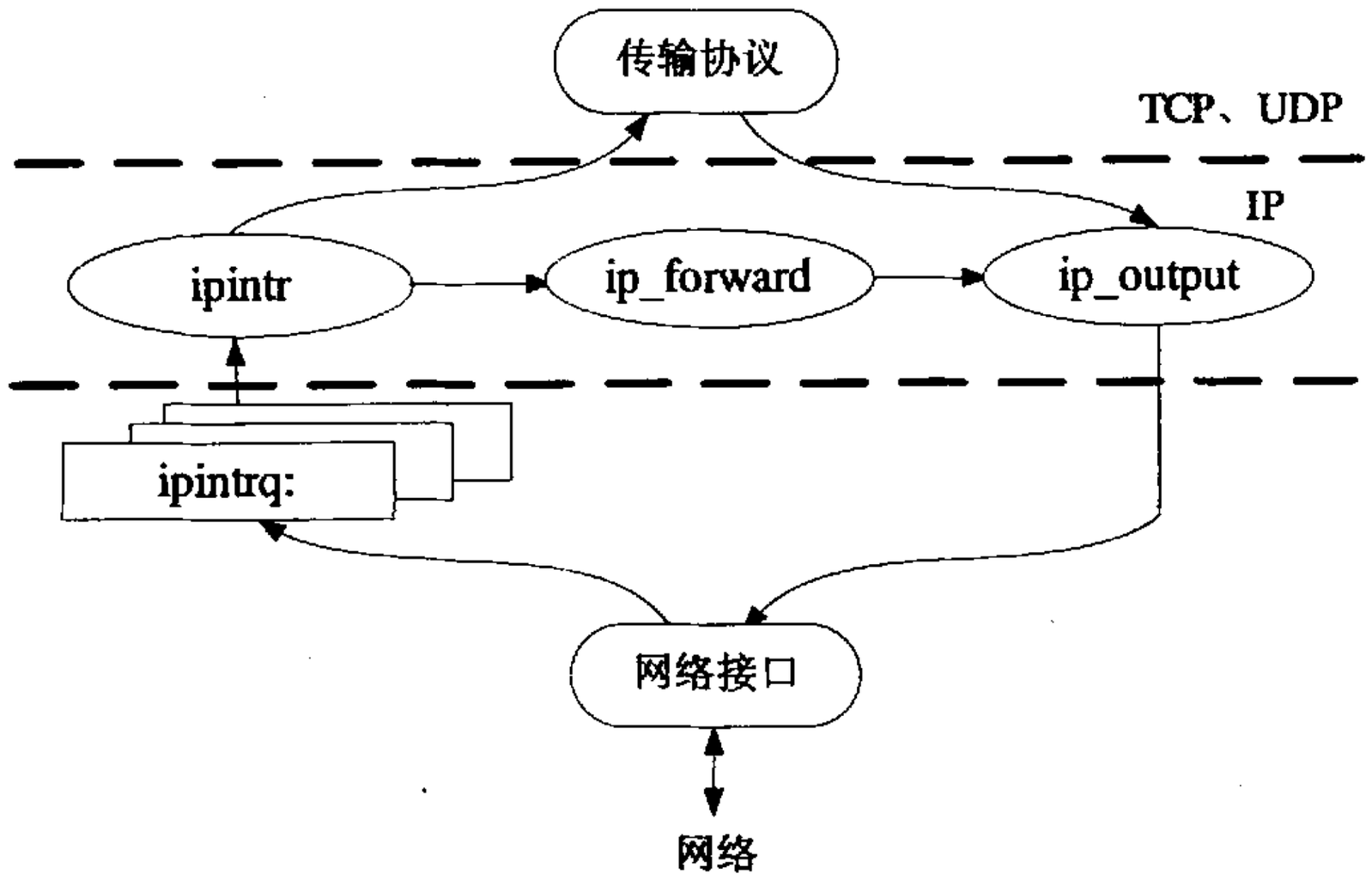


图 5-17　IPv4 协议栈的处理流程[5]

在软件中断处理中， ipintr 函数不断从 ipintrq 中移走和处理分组，直到队列为空。在最终的目的地，IP 把分组重装为数据报,并通过函数调用把该数据报直接传给适当的运输层协议。如果分组没有到达最后的目的地,对路由器来说,则 IP 把分组传给 ip_forward。传输协议和 ip_forward 把要输出的分组传给 ip_output，由 ip_output 完成 IP 首部、选择输出接口以及在必要时对分组分片。最终的分组被传给合适的网络接口输出函数。

可见，为了在 IP 层的 IPv4 协议栈中实现 IPSec 的支持，需要在 ip_input、ip_forward、ip_output 三个重要的输入、转发、输出函数中嵌入 IPSec 处理部分。

IPv6 协议栈的处理流程如下图 5-18 中所示：

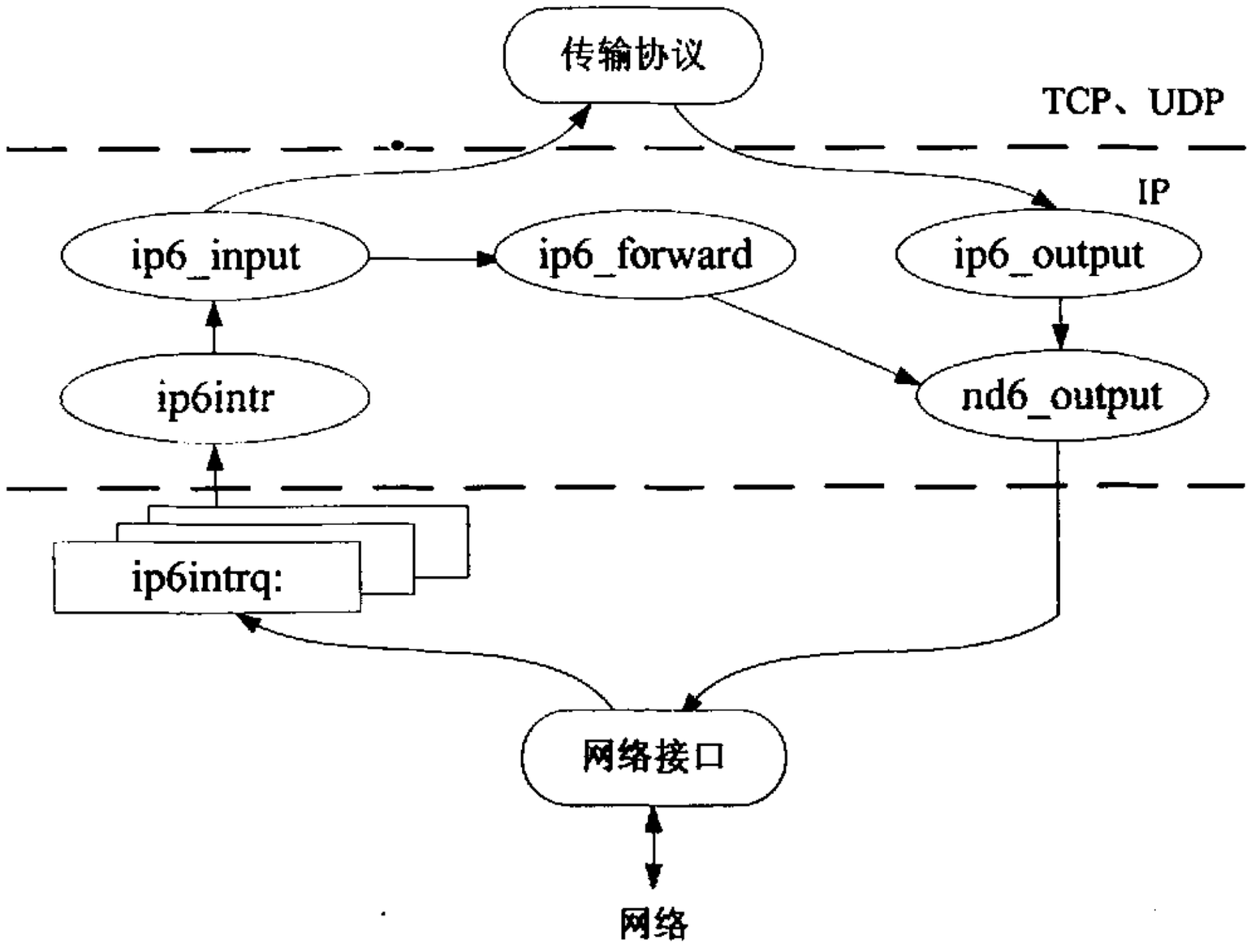


图 5-18　IPv6 协议栈的处理流程

IPv6 协议栈中的处理流程与 IPv4 的很相似，只是 IPv6 输入例程还需要调用 ip6_input 处理，而 ip6_forward 不再调用 ip6_output，而是直接调用 nd6_output 输出。同样，为了在 IP 层的 IPv6 协议栈中实现 IPSec 的支持，需要在 ip6_input、ip6_forward、ip6_output 三个重要的输入、转发、输出函数中嵌入 IPSec 处理。本着这样的思想，继续以下实现。

5.3.8.4 IPSec 输入的实现

1、IPv6 协议栈中的实现

IPSec 输入处理部分主要是在 ip6_input 函数中添加对 IPSec 协议的支持。ip6_input 函数的处理可以分为四个阶段：对到达分组验证、选项处理及转发、分组重装、分用。对于到达非最终目的地系统的分组需要在第二个阶段被转发，这是通过调用 ip6_forward 函数来实现。因此需要

在 ip6_forward 函数中添加 IPSec 处理部分以支持 IPSec 对转发包的处理：首先从输入的 mbuf 包中提取安全策略索引参数，然后调用 ipsec6_getpolicybyaddr 函数查找安全策略数据库 SPD，并对返回的策略进行合法性检查，根据安全策略决定对此包是进行丢弃、绕过还是安全处理——对丢弃的包直接释放 mbuf 链，对绕过的包将继续它的正常处理，对需要进行安全处理的包将会调用 IPSec 输出处理函数 ipsec6_output_tunnel，这在输出处理部分将会涉及到。

ip6_input 函数处理最后一阶段分用时，从 IPv6 包头中获取的下一个协议头被用 ip6_protox 数组映射到 inet6sw 数组的下标，ip6_input 调用选定的 protosw 结构中的 pr_input 函数来处理数据报包含的上层报文。但是在调用之前，需要先判断 protosw 结构中的 pr_flags 是否被标记为 PR_LASTHDR，这意味着将要处理最后一个协议头，那么就需要对此数据包先进行安全策略的验证，对于验证失败的包会被丢弃。在内核初始化时我们已经将 AH 和 ESP 协议注册到了 inet6domain，如果输入报文是含有 IPSec 安全头的数据包，那么所调用的 pr_input 函数就应该是 AH 或者 ESP 协议的输入处理函数。而且此部分协议分用是一个循环处理过程，直到所调用 pr_input 函数返回的下一个协议头被标记为 IPPROTO_DONE 时，才会继续处理 ip6intrq 中的下一个分组。

AH 的输入处理函数为 ah6_input，它的处理流程如下图 5-19 所示：ah6_input 的功能主要是对含有 AH 包头的数据包进行验证处理，它从接收到包头中提取有用信息构造 SADB 选择符，利用选择符查找 SADB 数据库，获得对应的 SA 条目，在 SA 有效的情况下，根据 SA 中给定的验证算法在 AH 验证算法表中查找对应的验证算法项，先进行抗重播检查，然后对整个数据包应用验证算法，并将获得的摘要同保存在 AH 头中的 ICV 值进行比较。如相符，IP 包就通过了身份验证；如不符，便丢弃该

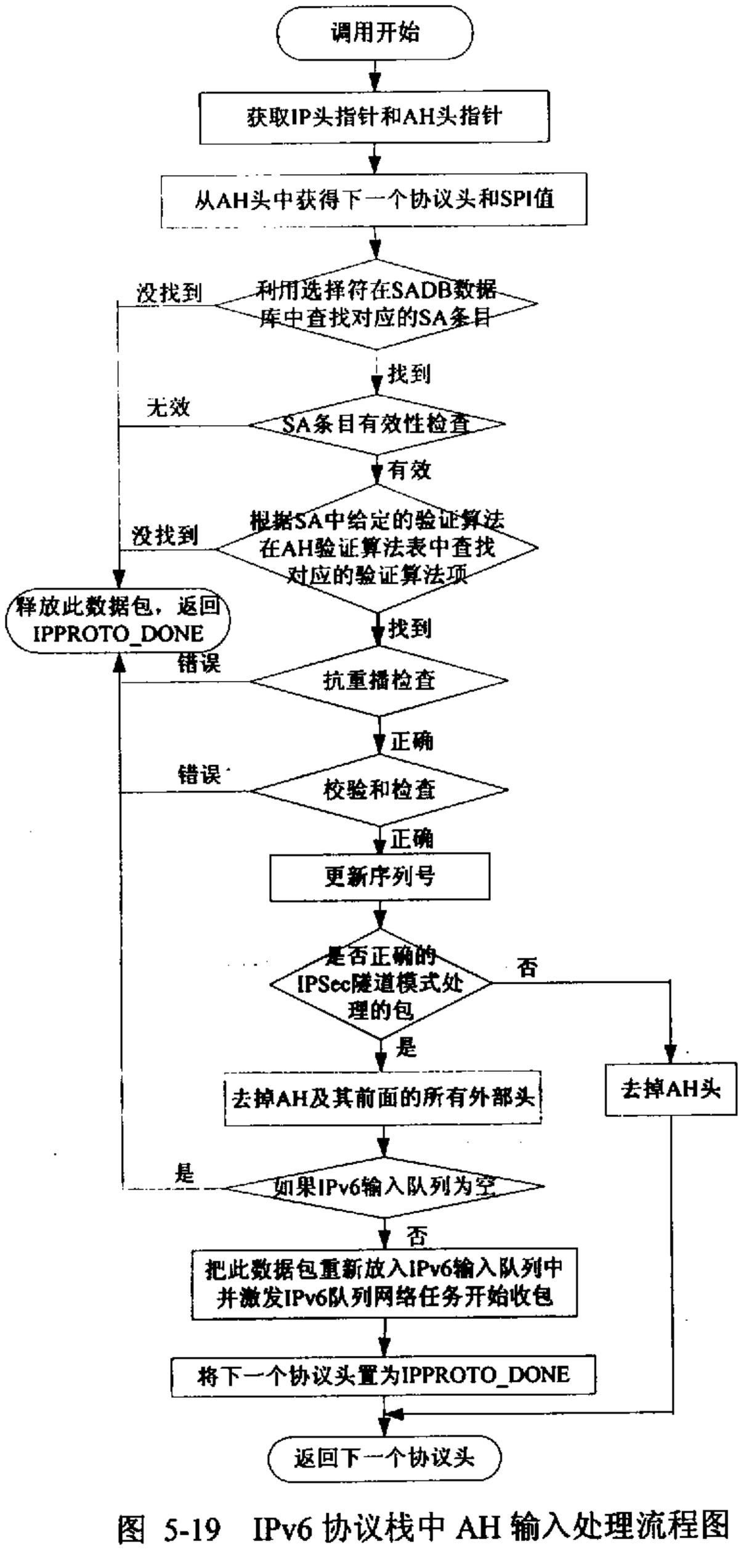


图 5-19　IPv6 协议栈中 AH 输入处理流程图

包。ICV 一经验证，如有必要，更新序列号，滑动接收窗口的序列号可以递增。如果它不是隧道模式处理的包，就直接去掉 AH 头交由上层协议处理，否则，需要将剥去 AH 头及其隧道头的 IP 包重新送回 IPv6 输入队列中等待再次处理。

ESP 的输入处理函数为 esp6_input，它的处理流程如下图 5-20 所示：

如果收到的 IPSec 包是一个分段，必须把它保留下来，直到这个包的其他部分收完为止。我们不能对一个不完整的 IPSec 包进行处理，因为可能会导致对它施行的初次校检失败——次数据完整性校检。

首先从接收到包头中提取有用信息构造 SADB 选择符，利用选择符查找 SADB 数据库，获得对应的 SA 条目，在 SA 有效的情况下，根据 SA 中给定的加密算法在 ESP 加密算法表中查找对应的加密算法项。查看所得 SA 中的验证算法字段是否存在，如果存在，接下来进行的便是对这个包进行身份验证，ESP 身份验证密文而不是明文。在验证的过程中，首先进行抗重播检查，接着利用恰当的密钥，把这个完整的 ESP 包（当然除开身份验证数据）传递到验证器那里（它取自 SA）。如果其结果能与“身份验证数据”字段中包含的数据相符（将身份验证算法可能需要的任何分段考虑在内），就可对这个包进行身份验证。接下来是解密。通过取自 SA 的密钥和密码算法，就可对 ESP 包进行解密，对正确解密的包需要去掉其 ESP 尾部。接着判断它是否是隧道模式处理的包，如果不是，就直接去掉 ESP 头及其 IV 值，返回协议头已处理完毕信息；否则，需要将剥去 ESP 头及其外部头的 IP 包重新送回 IPv6 输入队列中等待再次处理。

2、IPv4 协议栈中的实现

IPv4 协议栈的处理与 IPv6 协议栈的基本相同，不同之处就是 IP 输入例程 ipintr 直接处理 IP 层协议的输入，不需要再去调用别的函数，而

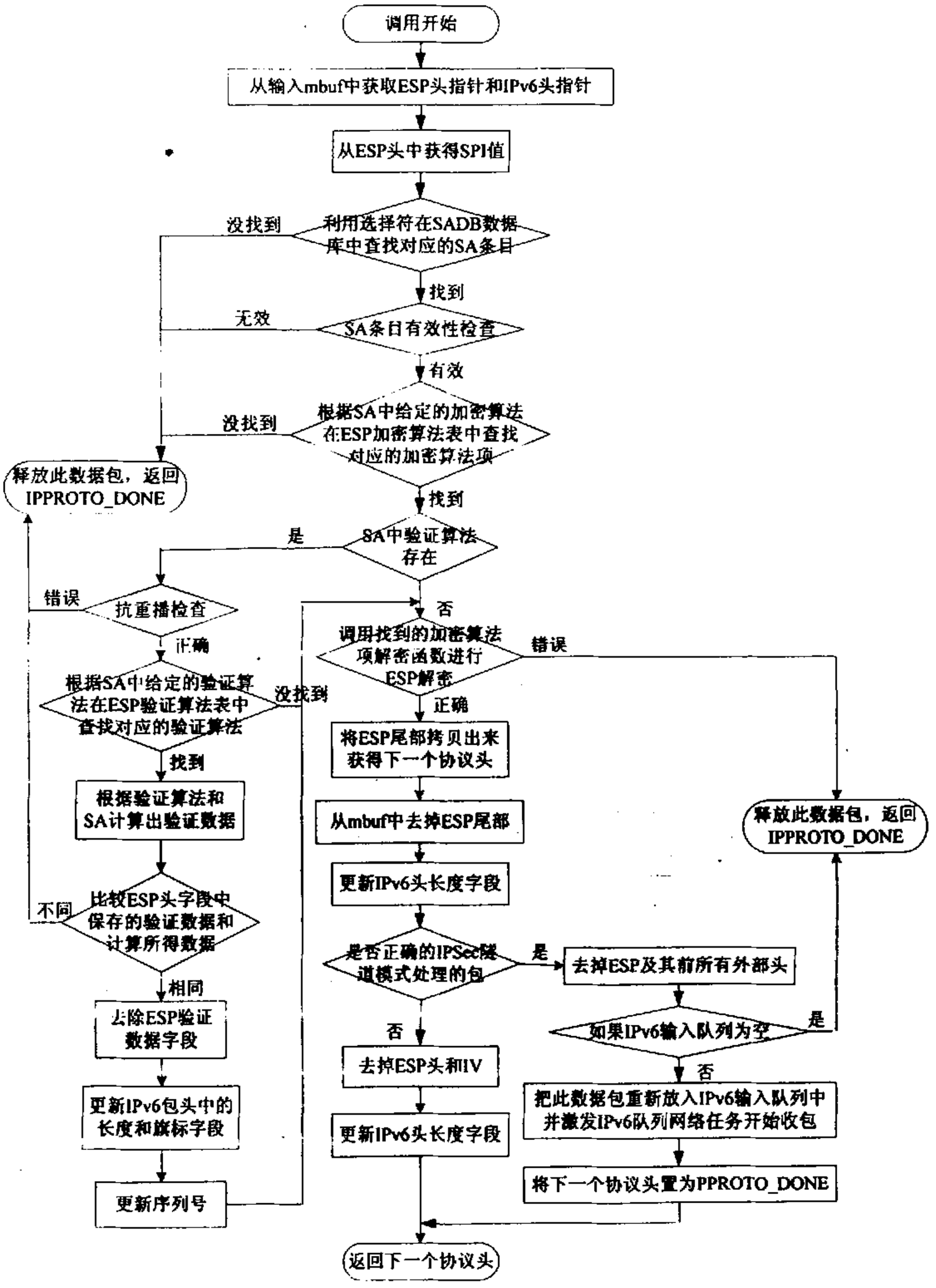


图 5-20　IPv6 协议栈中 ESP 输入处理流程图

ip_forward 将调用 ip_output 输出。故 IPSec 输入处理部分主要是在 ipintr 函数中添加对 IPSec 协议的支持。同 ip6_input 函数类似，ipintr 函数的处理也可以分为四个部分：对到达分组验证、选项处理及转发、分组重装、分用。在转发部分将会调用 ip_forward 函数进行包转发，因为 ip_forward 将调用 ip_output 处理，而 ip_output 中自有对 IPSec 输出处理部分，为避免功能重复，在 ip_forward 函数中不再有 IPSec 的具体处理，只是在最后计算隧道 MTU 时会进行判断，添加 IPSec 处理后安全包头长度。

ipintr 函数最后一部分分用时，数据报中指定的协议被 ip_p 用 ip_protox 数组映射到 inetsw 数组的下标，ipintr 调用选定的 protosw 结构中的 pr_input 函数来处理数据报包含的上层报文。但是在调用之前，也要先判断 protosw 结构中的 pr_flags 是否被标记为 PR_LASTHDR，这意味着将要处理最后一个协议头，那么就需要对此数据包先进行安全策略的验证，对于验证失败的包会被丢弃。在内核初始化时我们已经将 AH 和 ESP 协议注册到了 inetdomain，如果输入报文是含有 IPSec 安全头的数据包，那么所调用的 pr_input 函数就应该是 AH 或者 ESP 协议的输入处理函数。当 pr_input 返回时，ipintr 继续处理 ipintrq 中的下一个分组。其中 AH 和 ESP 的输入处理也与 IPv6 协议栈实现相似，在此不再赘述。

5.3.8.5 IPSec 输出的实现

1、IPv6 协议栈中的实现

与 IP 层输入处理不同，标准的 Internet 传输协议（TCP 和 UDP）直接调用 ip6_output，而不查询 inet6sw 表。对标准的传输协议而言，protosw 结构不必具有一般性，因为调用函数并不是在与协议无关的情况下接入 IP 的。IPSec 输出处理部分主要是在 ip6_output 函数中添加对 IPSec 协议的支持。ip6_output 函数中 IPSec 处理流程如下图 5-21 所示：

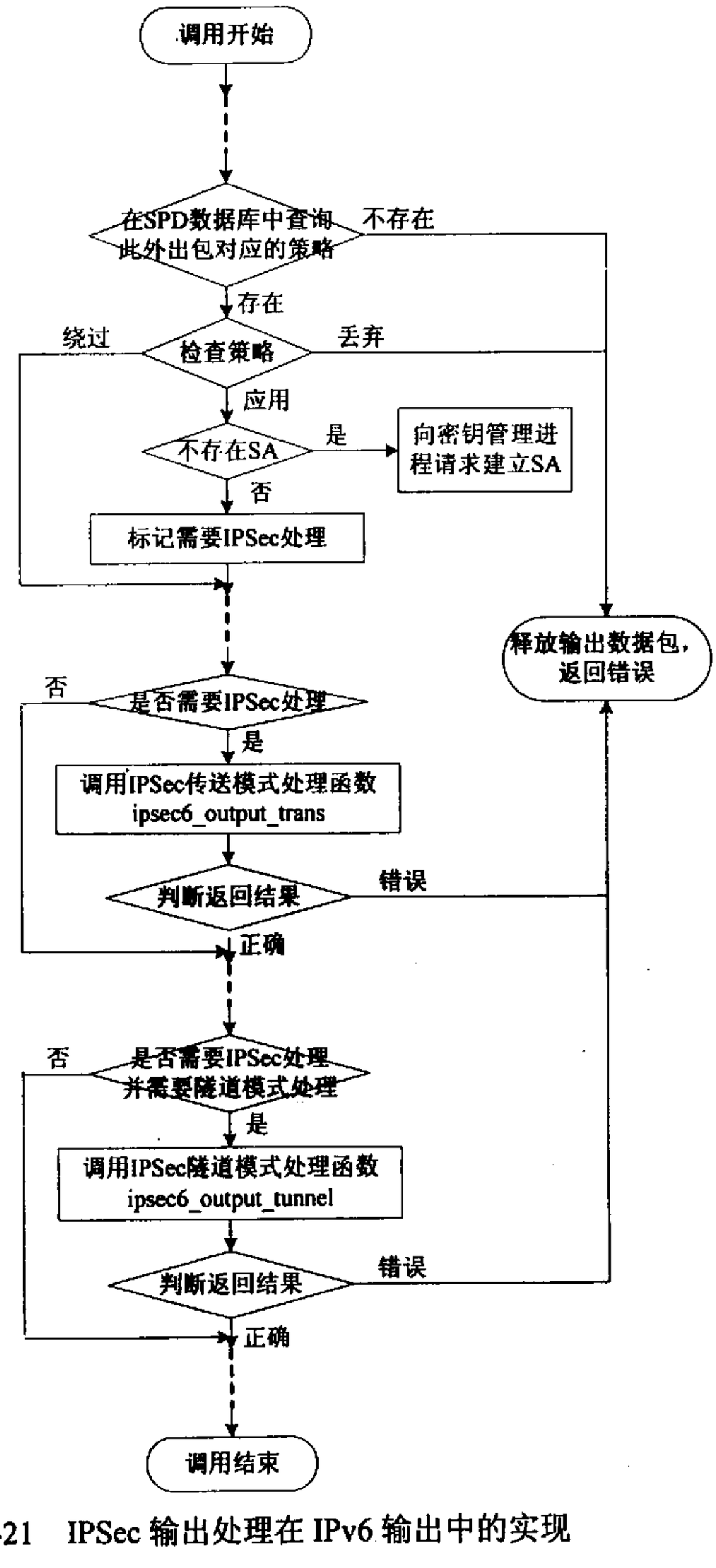


图 5-21　IPSec 输出处理在 IPv6 输出中的实现

其中，IPSec 传送模式处理函数 ipsec6_output_trans 的工作流程如图 5-22 所示：

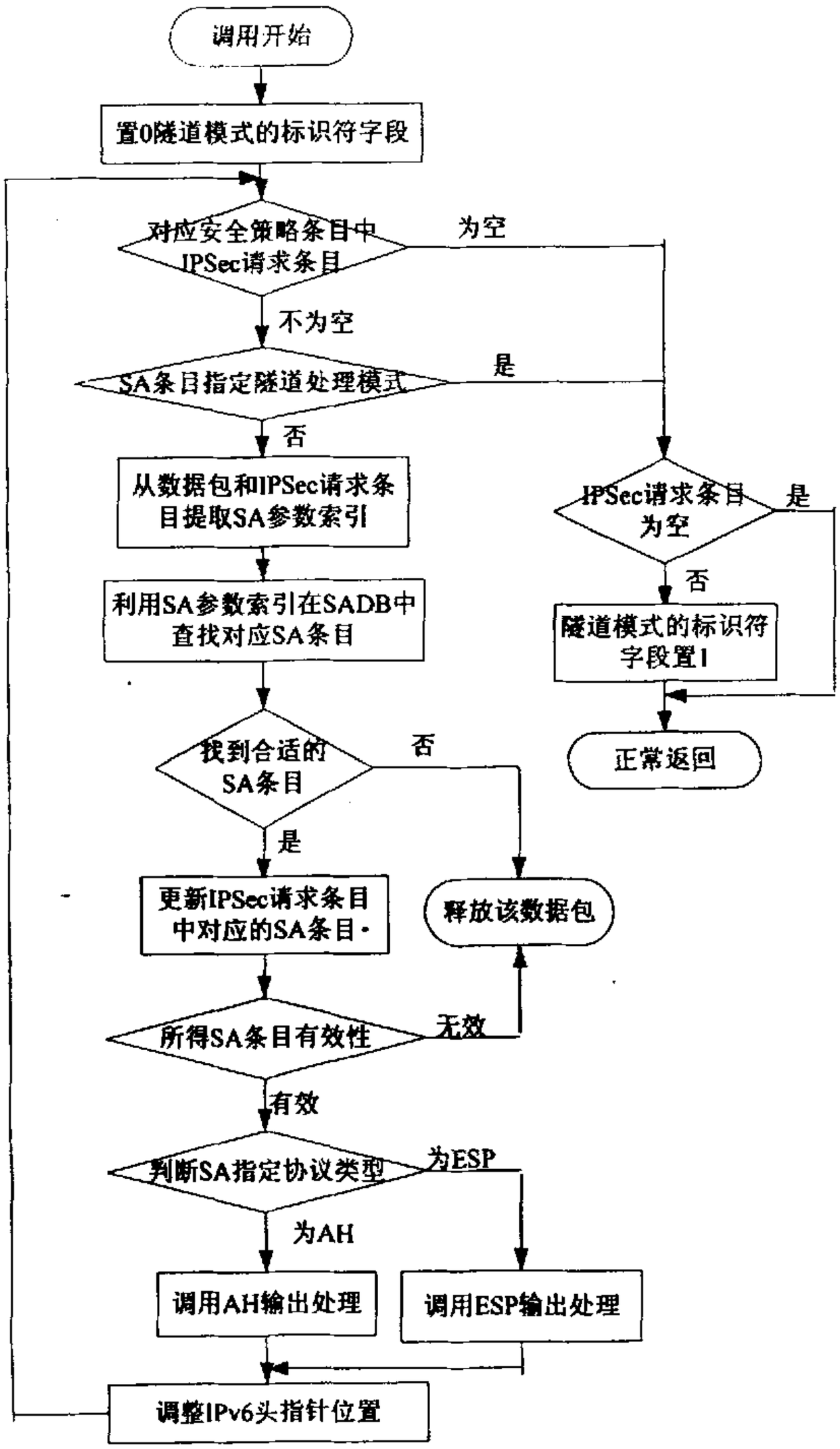


图 5-22　IPv6 协议栈中 IPSec 传送模式处理流程图

IPSec 隧道模式处理函数 ipsec6_output_tunnel 的工作流程如图 5-23 所示：

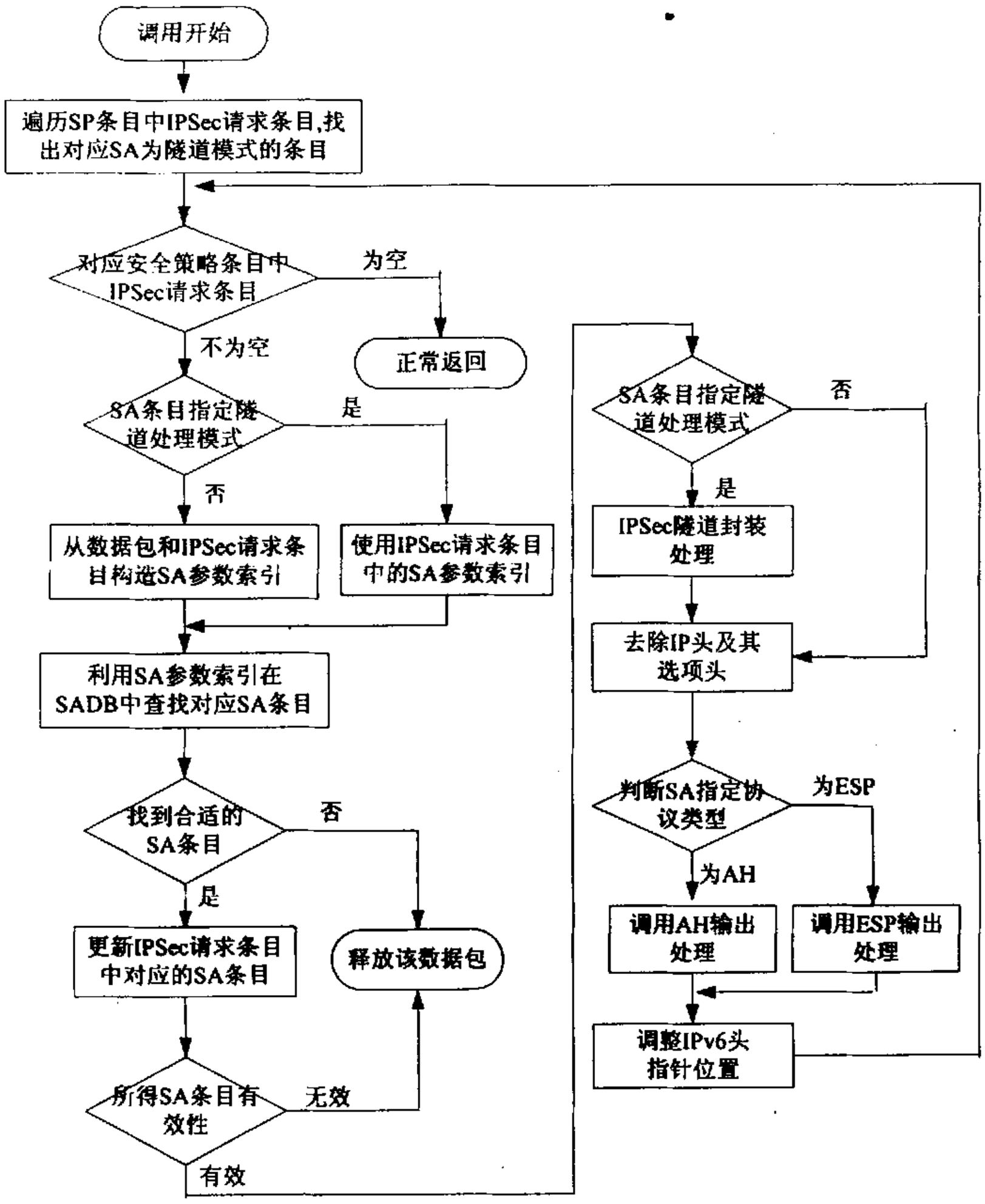


图 5-23　IPv6 协议栈中 IPSec 隧道模式处理流程图

在传送模式和隧道模式中调用的 AH 和 ESP 输出处理函数分别由 ah6_output 和 esp6_output 实现。

ah6_output 函数的主要功能就是对外出的数据包进行 AH 封装处理。它的处理流程如下图 5-24 中所示：

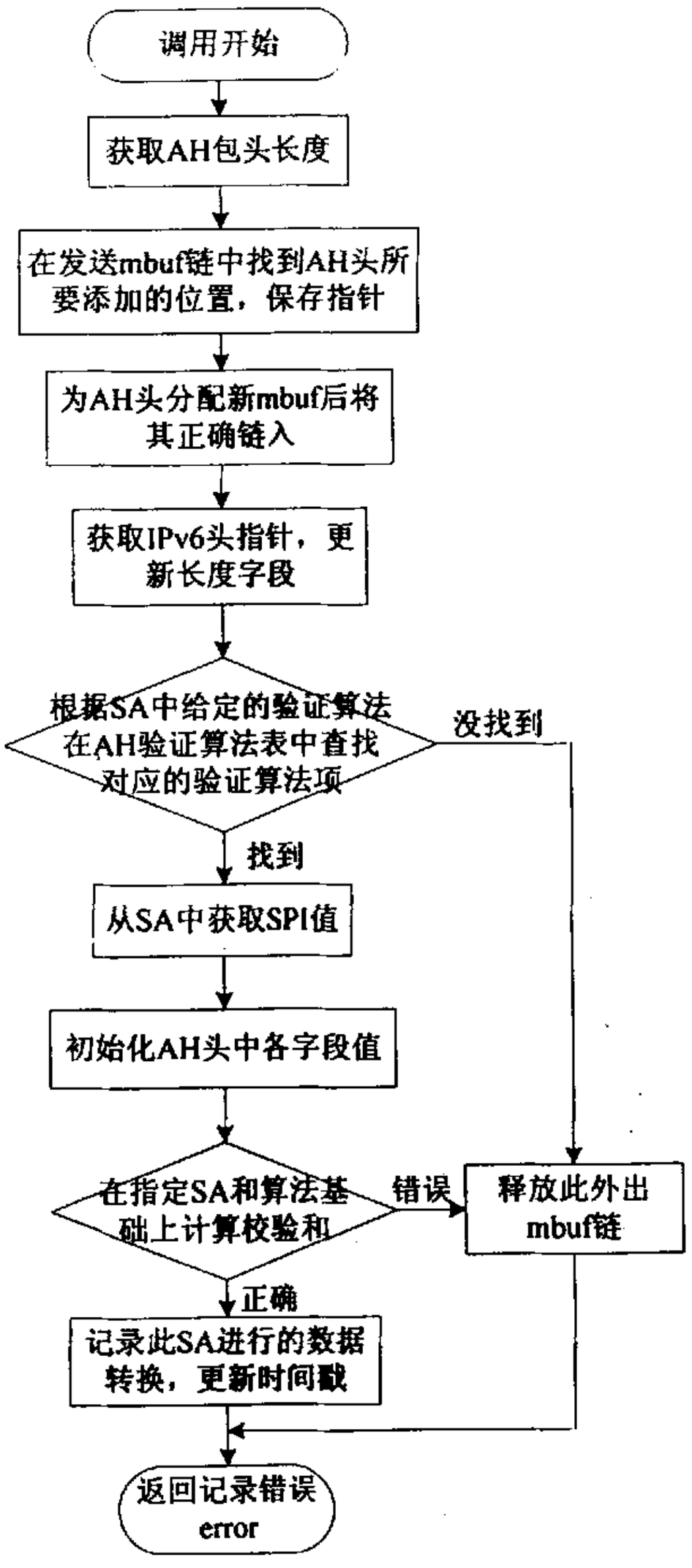


图 5-24　IPv6 协议栈中 AH 输出处理流程图

esp6_output 函数的主要功能就是对外出的数据包进行 ESP 封装处理。它的处理流程如下图 5-25 中所示：

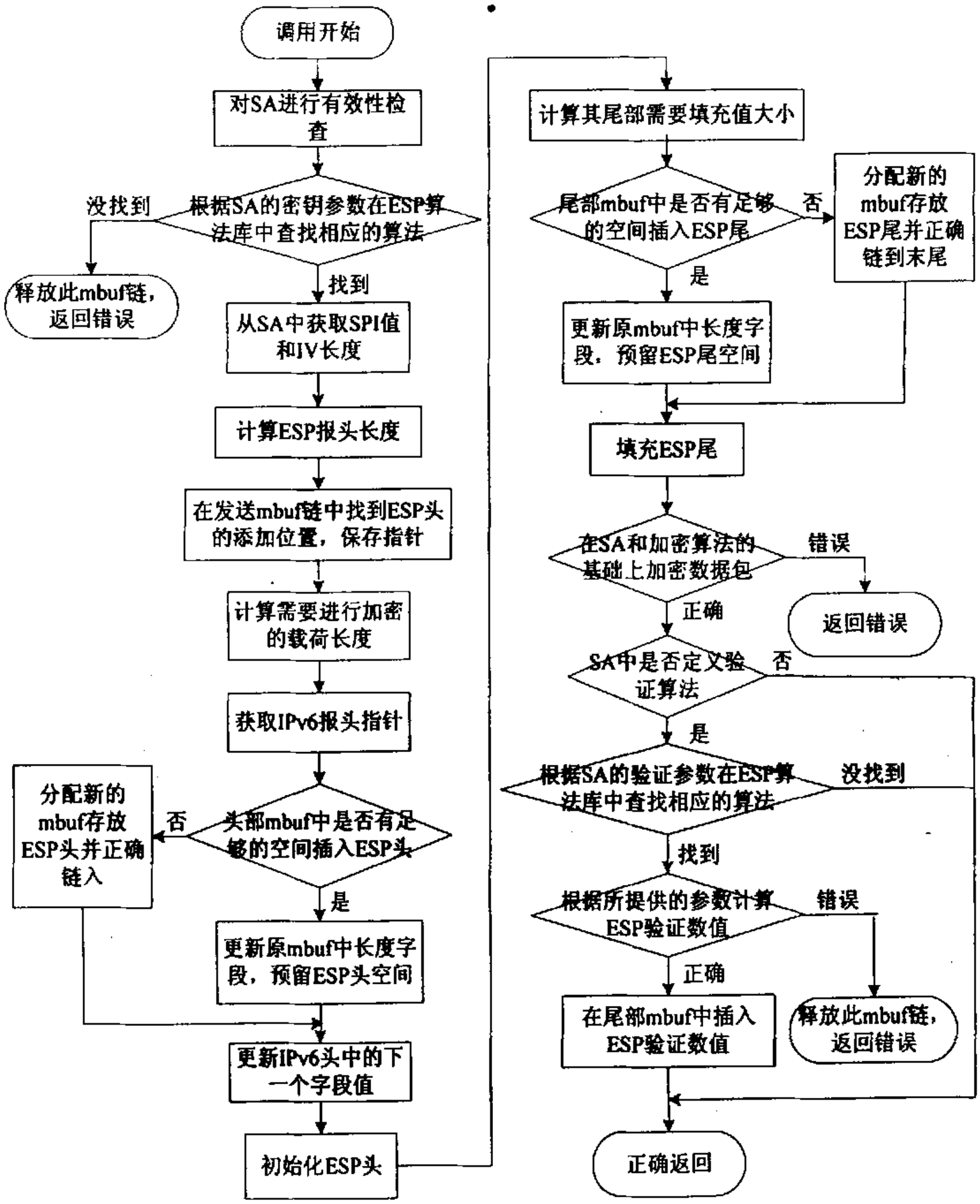


图 5-25　IPv6 协议栈中 ESP 输出处理流程图

2、IPv4 协议栈中的实现

IPSec 输出处理部分主要是在 ip_output 函数中添加对 IPSec 协议的支持：首先从输入的 mbuf 包中提取安全策略索引参数，然后调用 ipsec6_getpolicybyaddr 函数查找安全策略数据库 SPD，并对返回的策略进行合法性检查，根据安全策略决定对此包是进行丢弃、绕过还是安全处理——对丢弃的包直接释放 mbuf 链，对绕过的包将继续它的正常处理，对需要进行安全处理的包将会查看是否有对应的安全联盟存在，如果没有，将向密钥管理进程发起请求建立 SA。在这之后，会调用 IPSec 输出处理函数 ipsec4_output，由它来对外出的 IPv4 数据包进行安全处理。ipsec4_output 的处理流程如下图 5-26 所示，它在最后会根据 SA 所指定的类型来调用相应的协议处理，此处的 AH 和 ESP 协议输出处理与 IPv6 协议栈下的处理类似，在此不再赘述。

5.3.9　加密验证算法模块

5.3.9.1 功能描述

加密验证算法模块是一些标准算法处理的定义，主要是为 IPSec 处理过程中的加解密和验证运算提供密钥算法支持以及库函数调用。我们提供支持的验证算法包括 HMAC-MD5、HMAC-SHA1；加密算法包括 3DES -CBC。

5.3.9.2 主要数据结构

```
/*在内存中分配一个 MD5 的处理区*/
typedef struct
{
        u_int64_t   md5_count64;
        u_int8_t    md5_count8[8];
       u_int    md5_i;
       u_int8_t     md5_buf[MD5_BUFLEN];
} md5_ctxt;
```

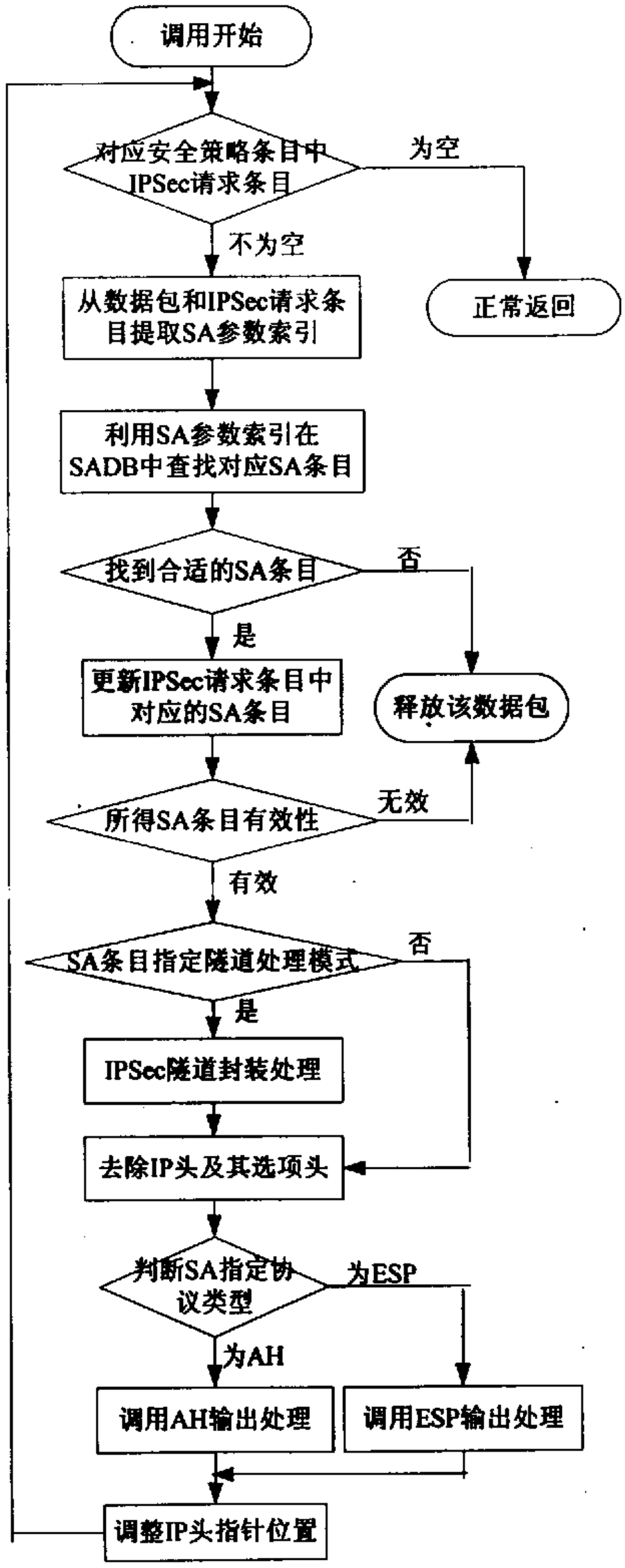


图 5-26　IPv4 协议栈中 IPSec 输出处理流程图

```
/*SHA1 算法处理区*/
struct sha1_ctxt {
        u_int8_t b8[64];
        u_int32_t    b32[16];
        u_int8_t count;
};

/*3DES 密钥的存贮格式*/
typedef struct des_ks_struct
{
        des_cblock _;
        DES_LONG pad[2];
} des_key_schedule[16];
```

5.3.9.3 验证算法的实现

1、MD5 算法的实现

程序的设计思想是，首先对消息按 512 位进行消息摘要计算直到最后一个分组，如果最后一个分组不足 521 位，则先进行填充再进行消息摘要计算；然后对消息的长度（64 位表示）进行消息摘要计算；最后将运算的结果进行级联就是 128 位消息摘要。

主要的实现函数有：初始化函数 MD5Init(x)；块处理函数 MD5Update(x, y, z)；综合处理函数 MD5Final(x, y)。

2、SHA 算法的实现

程序对需要处理的消息进行消息摘要运算，最后生成 160 位的消息摘要。具体思想是：首先对消息进行按 512 位进行消息摘要计算直到最后一个分组，如果最后一个分组不足 512 位，则进行填充再进行消息摘要计算；然后对消息的长度（64 位表示）进行消息摘要运算；最后将运算的结果进行级联就是最后的消息摘要。

主要的实现函数有：初始化函数 SHA1Init(x)；块处理函数 SHA1Update(x, y, z)；综合处理函数 SHA1Final(x, y)。

5.3.9.4 加密算法的实现

3、3DES 算法的实现

3DES 加密程序对需要处理的消息进行加密，它是一种分组对称加密算法，每 64 位为一个分组，对输入数据中的最后一个分组如果不是 64 位的，采用了后面全部补 0 的方法进行填充。具体思想就是，首先对分组中的数据进行初始置换，然后对转换后的数据按加密（使用密钥 K1）－解密（密钥 K2）－加密（使用密钥 K3）的顺序三次调用函数 des_encrypt2（对一个 64 为的数据进行 DES 加密解密运算的函数），最后将最终结果末置换。

主要的实现函数有：加密函数 esp_des_blockencrypt()，解密函数 esp_3des_blockdecrypt()。

5.4 基于实时 Linux 的 IPSec 设计实现[19]

本章 5.3 节详细介绍了基于 VxWorks 的 IPSec 设计实现，因为本课题中还涉及到了实时 Linux 下 IPSec 的实现，本节将针对 Linux 下的实现方式进行叙述。

虽然 VxWorks 和 Linux 是两个不同的操作系统，但都采用与 OS 集成的实施方式来实现IPSec，因此，我们仍然可以使用VxWorks下对IPSec模块划分思想的来进行 Linux 下的实现。其功能模块划分如图 5-4 中所示，各模块的功能不再赘述。

既然基于不同的操作系统，其内核差异颇大，各个模块的具体实现方法自然也不会相同，由于本论文的重点旨在讨论 VxWorks 下的实现操作，在本节中我们只针对 Linux 下 IPv4 和 IPv6 两个协议栈中 IPSec 的实现处理方式进行概要介绍。

5.4.1　IPv4 协议栈中的实现

由于 IPSec 在 IPv4 中不是需要强制实施的，因此我们在具体实现时尽量不直接修改协议栈源码，而是在其基础上添加了一套可供调用的专用处理程序。首先，我们注册了一个虚接口，将其与实际物理接口相绑定，并在 IP 内核路由表中添加一条指向虚拟接口的路由，使得需要经过安全处理的 IP 包都先被送入虚拟接口中，调用其驱动程序处理后再送往与之相绑定的实际物理接口。我们的 IPSec 的分组封装处理程序就放在虚拟设备的驱动程序中，从而使得不必直接修改 IP 模块的源码。

IPv4 协议栈中 IPSec 对 IP 数据包的接收、转发和发送处理的整个过程如图 5-27 所示。

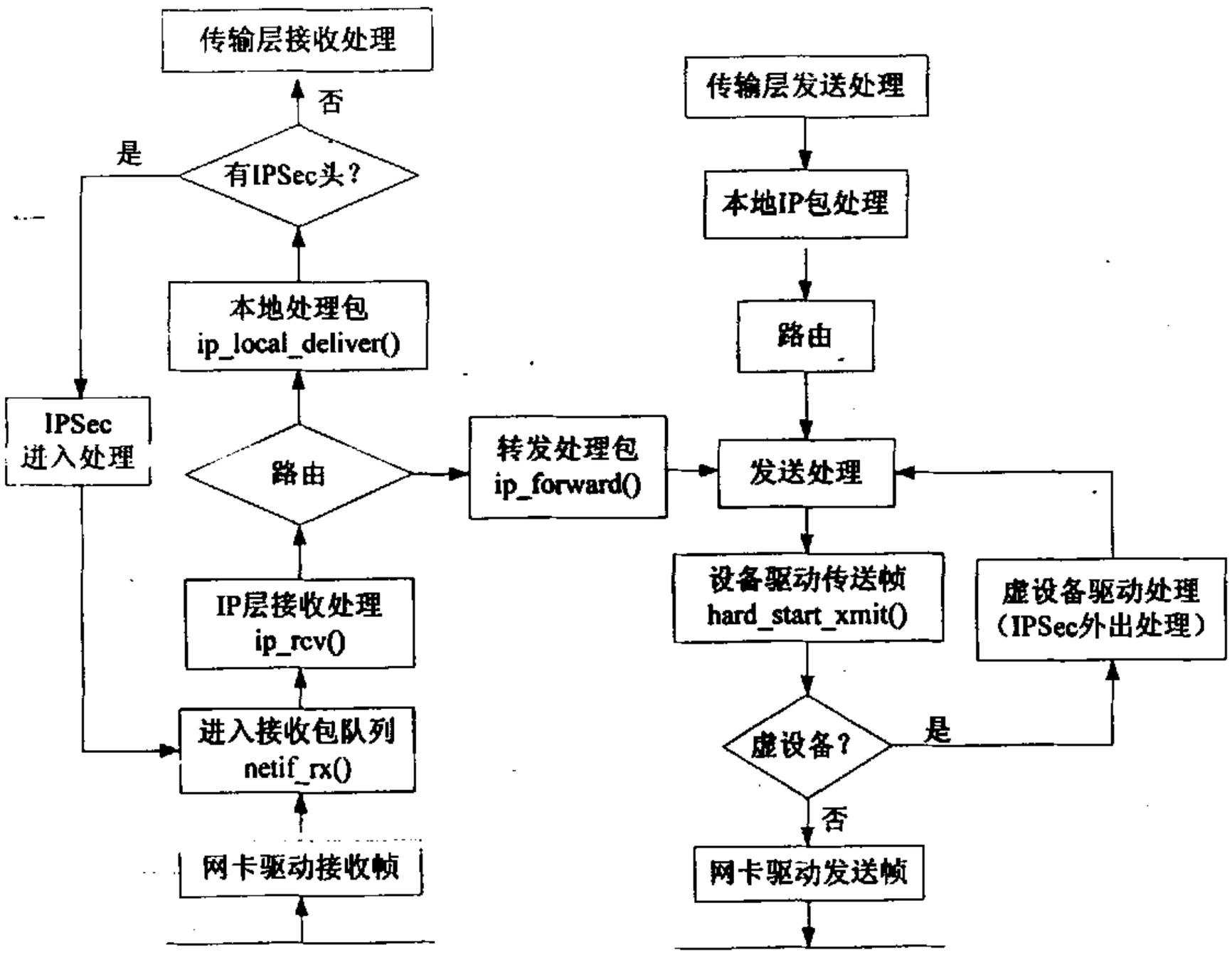


图 5-27　IPv4 协议栈中 IPSec 处理流程

当网卡驱动接收到链路层来的数据报时，通过软中断触发内核的中断处理程序，将网卡接收的数据报传送到内核空间，然后再通过 IP 层预处理程序将数据报转换为 IP 包。接着数据包将被送到路由处理模块，路由处理根据 IP 头的目的地址决定 IP 包是发送到本机还是继续转发。对于发送到本机的 IP 包，首先经过内核对 IP 包的处理，如：分片重组、选项处理等等，接着将会检查它的下一个包头，如果是一个非 IPSec 包头，将直接转交给传输层处理模块；如果发现是一个 IPSec 包头，就调用 IPSec 进入处理模块对 IP 包进行安全解封装：进行认证或解密等 IPSec 处理，并剥去 IPSec 头，之后再将此包送回进入接收包队列中重新路由处理。

对于进行转发处理和本地发送处理的 IP 包，如果不需要进行安全保护，经发送处理模块处理后，将被直接送到网卡发送出去；而需要进行安全处理的 IP 包在查找路由时将会找到一条指向虚接口的路由，那么它们在被送往网卡驱动发送之前将调用虚设备驱动，即 IPSec 外出处理模块进行发送处理：先是要决定它的安全处理策略，是丢弃、绕过还是进行 IPSec 处理，对需要 IPSec 处理的包进行认证或加密等。然后再将处理后的包送回协议栈的发送处理模块，由发送处理模块进行分片等处理，最后将包发送到网卡。

对应具体程序实现，对于 IPv4 的包，输入处理功能主要由 ipsec_rcv() 函数来完成。ip_local_deliver_finish()函数根据 hash 值（protocol = skb->nh.iph->protocol;hash = protocol & (MAX_INET_PROTOS - 1);）查找上层协议 ipprot = (struct inet_protocol *) inet_protos[hash]，如果发现上层协议是 AH 或 ESP，在调用上层协议处理函数 ret = ipprot->handler(skb) 时就调用 ipsec_rcv()（此函数指针在 AH 和 ESP 协议注册时进行赋值）。输出处理功能主要是由 ipsec_tunnel_start_xmit()函数来完成，当 IP 层调

用 Hard_start_xmit()向接口注册的处理程序发包时，如果发送接口是虚接口，在发送之前，调用此函数进行 IPSec 处理，在处理完之后，通过虚拟接口所绑定的物理接口发送出去。

5.4.2　IPv6 协议栈中的实现

IPSec 在 IPv6 中是强制实施的，因此我们不需要借助于虚拟设备驱动，可以通过直接修改协议栈的代码，将 IPSec 处理程序作为 HOOK（钩子）点，挂载到 IPv6 协议栈中。

IPSec 在 IPv6 协议栈中对 IP 数据包的处理过程如下图 5-28 所示。

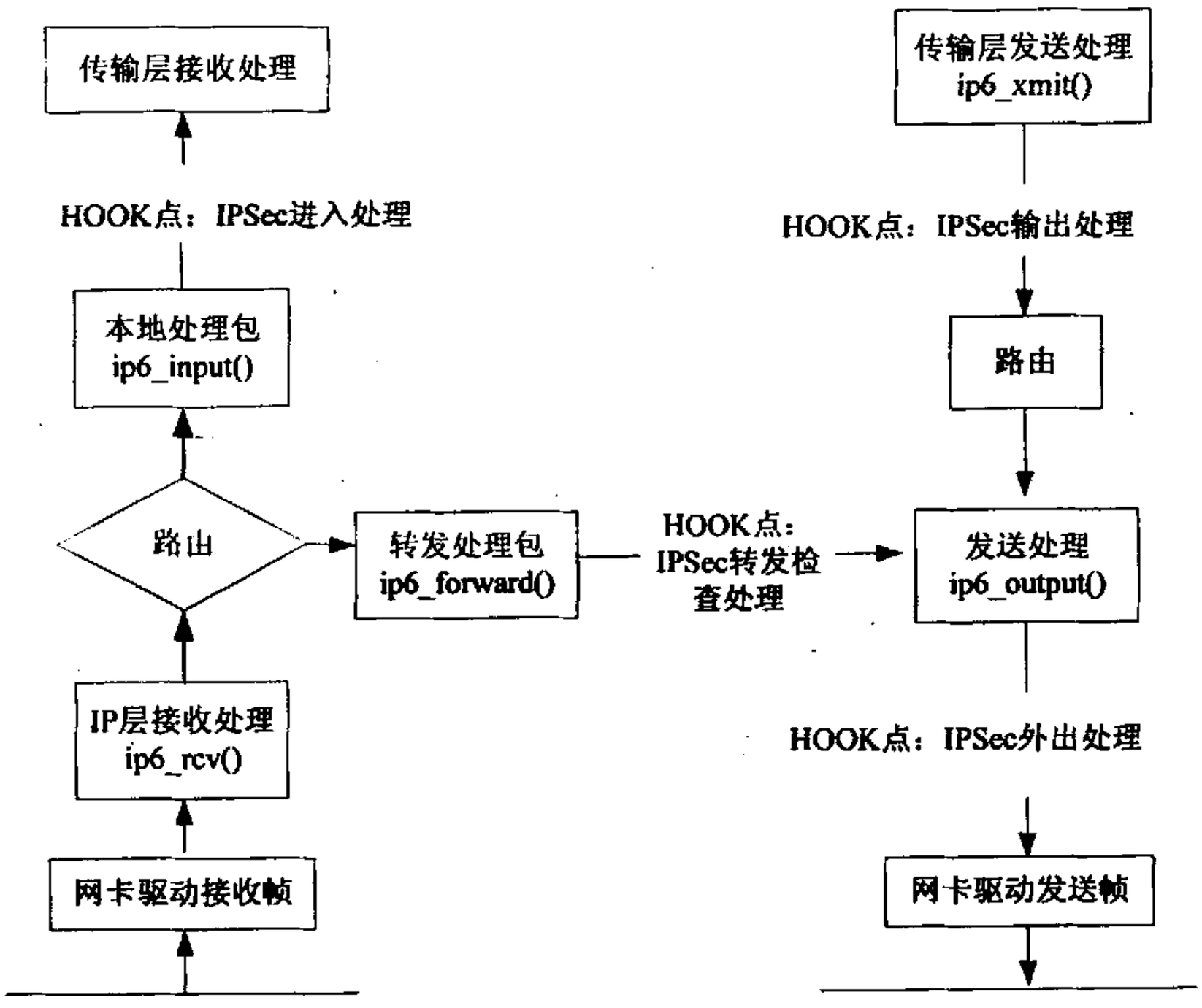


图 5-28　IPv6 协议栈中 IPSec 处理流程

IPv6 协议栈中网卡收包的过程与 IPv4 协议栈中的处理方式相似，对

发往本机的 IP 包，在进行本地处理后，将进入 IPSec 进入处理模块的 HOOK 点，由它来进行判断：对于非 IPSec 包，直接交给传输层模块进行处理，对于 IPSec 的包，就依据 SA（安全联盟）定义的策略进行相应的认证或解密处理后再转交给上层模块。

对于转发的 IP 包，首先进行转发处理：决定下一跳、减少 TTL、对某些特殊情况发送 ICMPv6 包。然后送交 IPSec 在转发处理模块中的 HOOK 点进行转发检查：是否存在经过安全处理的标志，如果存在，就认为它是一个 IPSec 包，直接交由下一步外出处理；反之，需要分析它的包头，并在 SPD（安全策略数据库）中查找其对应的处理策略，需要丢弃处理或者是策略不符的 IP 包将被丢弃，同时发送一个 ICMPv6 的通知报文，其它的 IP 包将转交协议栈的发送处理模块。

对于从传输层发送的报文，首先进行本地的 IP 包处理，生成 IP 包，接着就进入 IPSec 输出处理模块的 HOOK 点：IPSec 将为其查找 SPD 数据库，以决定它的相应处理策略。对于不需要处理的 IP 包，直接进行路由，决定 IP 包的出口；对于需要进行 IPSec 处理的 IP 包，将根据策略进行认证或加密等 IPSec 处理，在 IPv6 扩展头中插入 AH 头或者是 ESP 头，然后才对 IP 包进行路由。外出的 IP 包在经过协议栈的发送处理模块处理之后，再次进入 IPSec 的 HOOK 点：外出处理模块。在此被转发的 IP 包和本地发送的 IP 包都将经过发送前的最后一次检查：是否正确的进行了安全处理，发送正确处理的包，丢弃错误处理的包；是否需要经过隧道处理，如果是，添加外部 IPv6 头，否则绕过隧道处理。最后将包发送到网卡。

对应于具体实现时，输入处理功能主要由 do_ipsec6_in()函数来完成。在 ip6_iutput_finish()函数中，只要是在编译内核的配置选项里定义了宏 CONFIG_IPV6_IPSEC，都会调用此函数进行 IPSec 输入处理。输出处理

功能主要是由 do_ipsec6_out_tunnel ()函数来完成。在 ip6_output()函数处理开始之前，同样，只要是定义了宏 CONFIG_IPV6_IPSEC，都会调用此函数进行 IPSec 处理。

5.4.3　IPv4 与 IPv6 协议栈中实现的比较

可以很明显的看出，IPv6 协议栈中采用 HOOK 点的实现方式比 IPv4 协议栈中采用虚拟驱动的实现要有优越性，它不需要附加注册设备，也不需要为了 IPSec 的实施修改内核路由表，IPSec 自然融于协议栈中,有效地提高 IPSec 协议处理的效率，提高 IP 包在 IPSec 网关上的传送速度，使 IPSec 网关成为瓶颈的可能降低，也使得 IPSec 自身的安全性得到提高。但这种 HOOK 点的实现方案也存在它的不足：由于直接修改内核代码，系统的稳定性可能有所降低。因此在实现时必须充分考虑加入内核的 IPSec 模块对系统各个方面产生的影响。

第六章　系统调试与测试

本章介绍 VxWorks 下添加 IPSec 支持时对内核映象的调试与测试。分别介绍了调试和测试的方法，测试环境和测试结论。

6.1　调试

VxWorks 的系统开发平台 Tornado 环境我们在第三章中已经做了介绍，它采用主机--目标机交叉开发模型，应用程序在主机的 Windows 环境下编译链接生成可执行文件，下载到目标机，通过主机上的目标服务器与目标机上的目标代理程序的通信完成对应用程序的调测、分析。

在我们的调试环境中，我们以运行 Windows 2000 的 PC 机作为宿主机，以路由器的硬件板作为目标板，宿主机和目标板之间通过以太网相连。在搭建完上述调试环境以后，就可以进行在 VxWorks 内核添加 IPSec 支持功能的调试。下面将具体讲述此过程中的三个调试阶段。

调试是随着内核的开发同步进行的。在程序框架编写完成之后，即用 Torndao 的调试工具进行调试。这一阶段的调试目的主要是排除程序编写中的错误，确保程序能够正确的编译和运行。为了重新编译修改过的内核，在 Project 菜单下，选择相应硬件平台的生成 VxWorks 的命令，进行编译链接。如下图 6-1 中所示：

点击 OK 后，将进行 VxWorks 内核的编译，如果出现错误，将提示出错信息，我们就可以根据其出错信息对程序代码进行修改。内核的编译链接调试通过以后，就可以进行第二个阶段的调试：将 VxWorks 的内核映象通过 FTP 下载到目标板上。由于对内核进行了修改，所以在此步骤的下载过程中会出现一些错误，而 Tornado 提供的调试工具主要针对

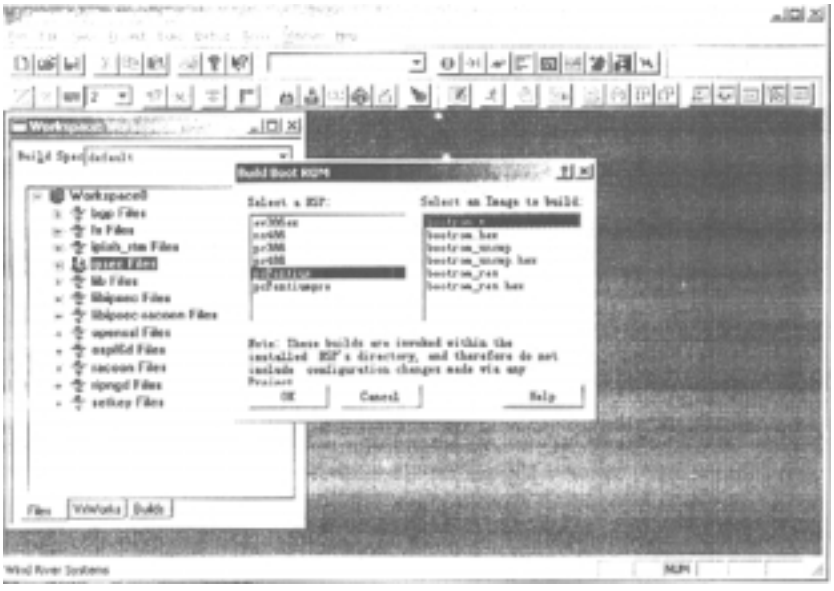


图 6-1 选择硬件平台重新编译 VxWorks 映象

应用程序的调试，对在内核映象下载过程中的错误无法显示，因此我们必须在内核中添加显示信息以跟踪发现错误。等到 VxWorks 内核映象顺利下载到目标板上，为了验证功能添加是否正确，能否正常运行，就需要第三个阶段的功能调试。此阶段的调试可以分为两个部分：套接口的调试和输入输出处理功能的调试。这个阶段的调试需要应用程序的配合，我们先在宿主机上启动 Target Server，通过以太网与目标板连接通信，然后启动宿主机上的 WindSh，WindSh 是一个驻留在主机内的 C 语言解释器，通过它可运行下载到目标机上的所有函数，同样 Tornado 开发环境也可以从 Shell 窗口下发命令和浏览。首先进行套接口的调试，为了验证在内核中添加的 PF_KEY 套接字能否工作，可以在 WindSh 下面执行以下命令：->socket(27,3,2)，其中第一个参数是协议族类型 PF_KEY，它的宏定义值为 27；第二个参数是套接字类型 SOCK_RAW，

它的宏定义值为 3；第三个参数是套接字协议 PF_KEY_V2，它的宏定义值为 2。如果此命令执行返回结果为 0，则表示套接口添加已经成功；否则，根据它返回的错误信息进行修改。接着我们利用手动密钥管理的方法来测试套接口与内核通信是否正常。在 Shell 界面下执行手工添加、删除、查询策略的命令，观察其返回结果与预定返回是否一致。对于内核输入输出处理的调试，首先要在内核中添加处理策略，然后再构造符合策略的数据包，在数据包发送接收处理过程中查看其处理结果，这种情况下，可以在内核中添加打印信息以跟踪调试。这三个阶段的调试都进行完毕以后，就可以开始测试了，我们下面将讲到具体测试环节。

6.2 测试

6.2.1 测试环境

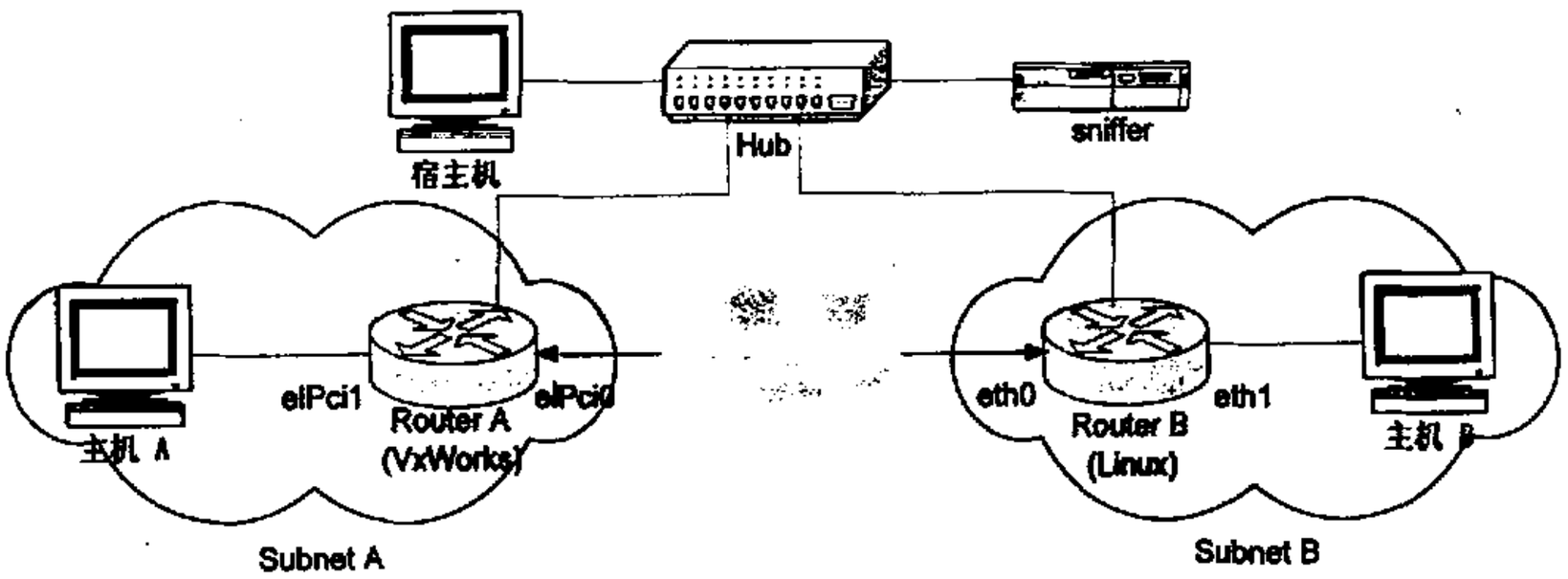


图 6-2　IPSec 测试网络拓扑结构图

为了对 IPSec 功能进行测试，我们构造了如图 6-2 所示的网络测试环境。图中共有两个子网：Subnet A 和 Subnet B，用以模仿中间穿越 Internet 网的情况。其中 Router A 是子网 A 的网关，运行 VxWorks 操作系统，它有一台宿主机相连，主机 A 是其子网内的一台主机；Router B 是子网

B 的网关，运行 Linux 操作系统，主机 B 是其子网内的一台主机。Router A 和 Router B 同时接在 Hub 上，以便于 sniffer 抓包分析协议。

6.2.2　测试内容

我们的测试内容是根据中科院《IPv6 Security 测试报告》而制定的，它包含以下内容：

1、验证在主机传送模式下能够对 IPv4/v6 数据包的 AH、ESP 输入输出进行正确处理。

2、验证在主机传送模式下能够对 IPv4/v6 数据包的 AH 和 ESP 结合输入输出进行正确处理。

3、验证在主机传送模式下能够按照 SPD 输入输出策略对 IPv4/v6 数据包的进行正确处理。

4、验证在路由器隧道模式下能够对 IPv4/v6 数据包的 AH、ESP 输入输出进行正确处理。

5、验证在路由器隧道模式下能够对 IPv4/v6 数据包的 AH 和 ESP 结合输入输出进行正确处理。

6、验证在路由器隧道模式下能够按照 SPD 输入输出策略对 IPv4/v6 数据包的进行正确处理。

其中，对 ESP 协议要求支持的加密算法有 3DES，验证算法有 HMAC-MD5、HMAC-SHA1；对 AH 协议要求支持的验证算法有 HMAC-MD5、HMAC-SHA1。

6.2.3　测试结果

如图 6-2，我们在两台路由器 Router A 和 Router B 上实施 IPSec，可以将它们分别配置为传送模式和隧道模式：当配置为传送模式时，IPSec 仅保护两台路由器之间的通信；当配置为隧道模式时，IPSec 仅保护路

B 的网关，运行 Linux 操作系统，主机 B 是其子网内的一台主机。Router A 和 Router B 同时接在 Hub 上，以便于 sniffer 抓包分析协议。

6.2.2 测试内容

我们的测试内容是根据中科院《IPv6 Security 测试报告》而制定的，它包含以下内容：

1、验证在主机传送模式下能够对 IPv4/v6 数据包的 AH、ESP 输入输出进行正确处理。

2、验证在主机传送模式下能够对 IPv4/v6 数据包的 AH 和 ESP 结合输入输出进行正确处理。

3、验证在主机传送模式下能够按照 SPD 输入输出策略对 IPv4/v6 数据包的进行正确处理。

4、验证在路由器隧道模式下能够对 IPv4/v6 数据包的 AH、ESP 输入输出进行正确处理。

5、验证在路由器隧道模式下能够对 IPv4/v6 数据包的 AH 和 ESP 结合输入输出进行正确处理。

6、验证在路由器隧道模式下能够按照 SPD 输入输出策略对 IPv4/v6 数据包的进行正确处理。

其中，对 ESP 协议要求支持的加密算法有 3DES，验证算法有 HMAC-MD5、HMAC-SHA1；对 AH 协议要求支持的验证算法有 HMAC-MD5、HMAC-SHA1。

6.2.3 测试结果

如图 6-2，我们在两台路由器 Router A 和 Router B 上实施 IPSec，可以将它们分别配置为传送模式和隧道模式：当配置为传送模式时，IPSec 仅保护两台路由器之间的通信；当配置为隧道模式时，IPSec 仅保护路

由器所在子网之间的通信。下图 6-3 与图 6-4 分别显示了 IPv4 协议栈与 IPv6 协议栈下经过 IPSec 加密保护的数据包。

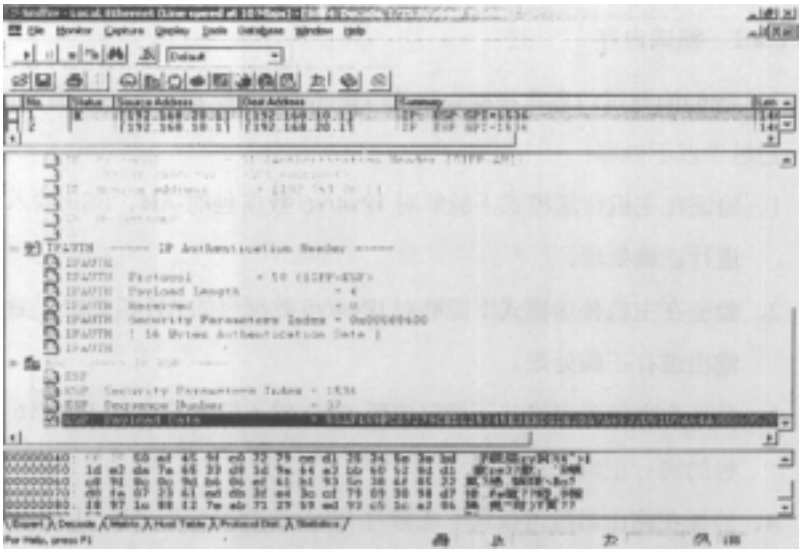


图 6-3　IPSec 保护的 IPv4 数据包

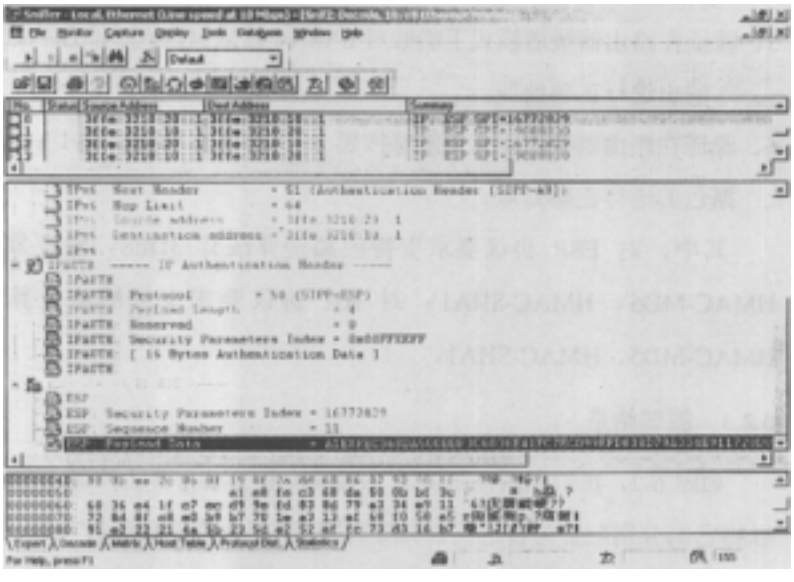


图 6-4　IPSec 保护的 IPv6 数据包

第七章 总结和展望

总结了本文所做工作，对 IPSec 协议提出了一些改进建议，并展望了在此基础上可继续深入研究的工作。

7.1 总结

随着 Internet 应用的日益广泛，网络安全问题也逐渐成为人们关注的焦点。据此，IETF 提出了新一代 IP 网络安全协议 IPSec，用以为 IPv4 和 IPv6 提供可互操作的、高质量的基于密码的安全性。本文系统综述了 IPSec 产生的背景、优势以及发展趋势，介绍了国内外 IPSec 技术的研究现状，论述了研究和实现 IPSec 协议的重要意义。

经过业界多年的努力，IPSec 协议不断得到增强和完善。本文深入探讨了 IPSec 协议的基本知识，包括 IPSec 的实施、模式、系统构成、组成协议等等，为以后具体的设计实现做了良好的铺垫。

做为 IP 网络安全协议，IPSec 在 Internet 上的应用极为广泛，已经逐渐成为构建虚拟专用网的基础。而它在实际应用中的主要实施正是在网络节点——路由器上。本文结合实验室路由器协议栈软件的课题，在自主研发的路由器协议栈软件上对 IPSec 的设计提出了自己的方案：作为路由器协议栈软件功能模块实现，采用与操作系统内核集成实施的方式，将 IPSec 处理与 IP 层融为一体，同属内置功能，使软件开发方便，性能稳定，兼容性好。

随着互联网的蓬勃发展，新应用层出不穷。原有 IPv4 网络已经不能从根本上满足人们对网络地址空间、QoS、移动性以及安全性等方面的更高要求。IPv6 技术的发展成为必然。但是由于种种原因，导致 IPv6

发展缓慢，针对目前网络仍是以 IPv4 网络构架为主，IPv6 网络处于实验床阶段的现状，分别设计实现了 IPv4 和 IPv6 协议栈下的 IPSec，以适应从 IPv4 向 IPv6 过渡阶段的需求状况。

此外，从安全性角度考虑，在实现 IPSec 时采用现有的多种加密算法，对所在的计算环境安全依赖小，产生的信息泄露少，同时，从可扩展方面考虑，加密验证算法被单独模块化设计，可以方便的添加新的算法，而且可以实现与新的算法无缝连接。

7.3 展望

IPSec 作为网络安全的一个主要实现标准，有关其功能的扩展性及其运行性能成为其研究重点。本文作者在自主开发的路由器协议栈上实现了 IPSec 协议族，从而实现了路由器的加密安全特性，但在研究工作中仍然面临一些有待进一步完善、发展的工作目标，其主要体现在以下几个方面：

1、IPSec 处理对路由器性能的影响

由于引入了 IPSec，使路由器转发延迟的变大，其中的性能瓶颈就是应用于数据包的加密处理。因为目前我们使用的加密算法都是通过软件实现的，尽管已经对加密算法进行了优化，然而使用加密还是给 CPU 增加了很多负载，这对路由器的整体性能影响非常大。而使用硬件加密会对我们的加密性能产生很好的影响，同时也能减轻 CPU 的负载，提高路由器的性能，这是以后研究的一个方向。

此外，由于每个数据包的进出都需要查找 SPD 数据库以确定其应用的策略，而且需要安全处理的包还要查找 SADB 数据库来获得对应安全策略的具体参数，因此，对 SPD 和 SADB 数据库查找的开销也将制约

着路由器的处理速度。合理组织这两个数据库的存储结构，采用高效的查找算法也将是我们努力的一个方面。

2、IPSec 与 NAT 的共存问题

IPSec 和网络地址转换协议 NAT（Net Address Translation）在路由器上的应用都已经十分的广泛了，然而，由于 IPSec 协议架构本身特点以及缺乏支持 IPSec 的 NAT 设备，当 IPSec 和 NAT 一起运行时就会出现很多问题：AH 在传送模式和隧道模式下均对整个 IP 包进行了认证，认证部分覆盖了 NAT 设备对 IP 头地址字段的修改，这将导致接收端对数据包完整性认证的失败。另外，在传送模式下，ESP IPSec 不能和 NAPT 一起工作，因为在这种传送模式下，端口号受到 ESP 的保护，端口号的任何改变都会被认为是破坏。在隧道模式的 ESP 情况下，TCP/UDP 报头是不可见的，因此不能被用于进行内外地址的转换，而此时静态 NAT 和 ESP IPSec 可以一起工作，因为只有 IP 地址要进行转换，对高层协议没有影响。

3、IPSec 应用于组播协议

随着 Internet 上许多新的高速宽带多媒体应用的产生，组播技术应运而生，它在路由器上的应用需求也逐渐增大，其安全性问题也日益受到重视。人们为单播系统的安全提出了很多解决的方案，而且制定了一些标准。但是对于组播系统的安全性来说，还存在许多尚未解决的问题。我们提出将 IPSec 应用于组播协议，以保证组播的安全性，因为 IPSec 提供了一个遍布 Internet 的切实可行的安全协议套件，它在单播系统的安全实施中已经得到了很好的应用。使用现存协议，不论从系统设计者的观点还是从用户的观点来看，都可以给安全组播的应用带来很多好处。但这种设计也存在一定的负面影响，因为 IPSec 协议本身是为单播系统设计的，如果应用于组播，就迫使我们除了要解决好与现行实施的

兼容性问题外，还要对现有 IPSec 协议进行多方面的改进，提出一种合理可行的解决方案，这些都有待进一步的研究和实践。

4、与移动 IP 相结合

目前移动 IP 也是研究的热门话题，移动 IPv6 的设计，除了能够满足节点的移动性外，还应保障通信的安全，至少安全状况不比现在的 IPv4 更差。但移动计算与普通计算的环境及特性存在较大区别，如多数情况移动计算是在无线环境下，且移动节点需要不断更改通信地址(称之为转交地址)，这些都会导致许多安全问题，也给 IPSec 带来了新的挑战。

参考资料

[1] 张宏科，张思东，刘文红著 ，《路由器原理与技术》，北京：国防工业出版社，2003。

[2] Carlton R.Davis 著，周永彬、冯登国等译，《IPSec ：VPN 的安全实施》，清华大学出版社，2002。

[3] Naganand Doraswamy, Dan Harkins 著，京京工作室译，《IPSec 新一代因特网安全标准》机械工业出版社，1999 年 12 月。

[4] W.Richard Stevens 著，范建华等译，《TCP/IP 详解 卷 1：协议》，机械工业出版社，1999。

[5] W.Richard Stevens 著，陆雪莹等译，《TCP/IP 详解 卷 2：实现》，机械工业出版社，2000。

[6] W.Richard Steens 著，施振川等译，《Unix Nerworking Programming》（第 1 卷）,清华大学出版社，1999。

[7] W.Richard Steens 著，杨继张译，《Unix Nerworking Programming》（第 2 卷）,清华大学出版社，2000。

[8] 孔祥营，柏桂枝著，《嵌入式实时操作系统 VxWorks 及其开发环境 Tornado》，中国电力出版社，2002。

[9] 唐寅著，《实时操作系统应用开发指南》，中国电力出版社，2002。

[10] 王欲静，《IP 安全性与 IPSec 协议的研究》。

[11] 王蔚，冯运波等，《IPSec 的实现途径及主要产品》。

[12] 王作芬，王芙蓉等，《PF_KEY 及其在 IPSec 中的实现》。

[13] 商超，《基于 IPSec 的 VPN 技术研究与实现》。

[14] 张晓华，李智涛等，《VxWorks 网络协议栈的 MUX 接口》。

[15] 覃海峰，《VxWorks 环境下远程接入服务器内核的研究设计》。

[16] 朱蕾，张勇等，《基于 IPSec 的安全路由器设计与实现》。

[17] 电子产品世界，《嵌入式网络软件在低端路由器中的研究与实现》。

[18] 宗瑞锐，刘文红，张宏科，《基于 IPSec 的组播的研究和实现》，北京交通大学学报，2003.12。

[19] 宗瑞锐，张宏科，《IPv4 及 IPv6 中的 IPSec 实现》，计算机工程与应用，2004.2。

[20] 宗瑞锐，张宏科，《基于 IPv6 的 IPSec 研究与应用》，电信快报，2003.12。

[21] Wind River Systems,"VxWorks Network Protocol Toolkit Programmer's Guide 5.4",2001.

[22] Wind River Systems,"VxWorks Networking Programmer's Guide"。

[23] Wind River Systems, "VxWorks Programmer's Guide"。

[24] S. Kent, R. Atkinson,"Security Architecture for the Internet Protocol" RFC2401, 1998.11

[25] S. Kent, R. Atkinson, "IP Authentication Header(AH)", RFC2402

[26] C. Madson, R. Glenn,"The Use of HMAC-MD5-96 within ESP and AH", RFC 2403,1998.11

[27] C. Madson, R. Glenn, "The Use of HMAC-SHA-1-96 within ESP and AH", RFC 2404,1998.11

[28] C. Madson, N. Doraswamy, "The ESP DES-CBC Cipher Algorithm With Explicit IV",RFC2405,1998.11

[29] S. Kent, R. Atkinson,"IP Encapsulating Security Payload (ESP)", RFC2406, 1998.11

[30] D. Piper ,"The Internet IP Security Domain of Interpretation for

ISAKMP", RFC 2407,1998.11

[31] D. Maughham, M.Schertler,etc."Internet Security Association and Key Management Protocol (ISAKMP)", RFC2408,1998.11

[32] D. Harkins, D. Carrel, "The Internet Key Exchange", RFC2409, 1998.11

[33] R. Glenn, S. Kent "The NULL Encryption Algorithm and Its Use With IPSec", RFC 2410,1998.11

[34] R. Thayer, N. Doraswamy, R. Glenn, "IP Security Document Roadmap",RFC2411, 1998.11

[35] H. Orman, " The OAKLEY Key Determination Protocol",RFC2412, 1998.11

[36] R. Pereira, R. Adams, "The ESP CBC-Mode Cipher Algorithms",RFC2451,1998.11

[37] M.Oehler,R.Glenn, "HMAC-MD5 IP Authentication with Replay Prevention" ,RFC 2085,1997.2

[38] C. Metz, B. Phan, "PF_KEY Key Management API, Version 2" RFC2367, 1998.7

[39] A. Keromytis, N. Provos, "The Use of HMAC-RIPEMD-160-96 within ESP and AH",RFC2857,2000.6

[40] H. Krawczyk, M. Bellare, R. Canetti. "HMAC: Keyed-Hashing for Message Authentication"RFC2104,1997.2

致 谢

作为我研究生科研工作和学习的总结，这篇论文无疑凝结着我多年求学的思索和努力，同样也蕴涵着我的亲人、老师和朋友的支持与帮助。

首先感谢我的导师张宏科教授。张老师作为在通信领域知名的中青年专家教授，在学习和科研实践中，不但给我指明了前进的方向，鞭策我们勇于面对挑战、勤钻研、多思考、多交流，还从具体的问题上给我以指导和帮助，帮助我理清概念，扩展思路，发现和解决实际问题，其严谨的治学态度和求实的科学精神使我终生受益。此外张老师还教会了我许多做人的道理，要求我们树立正确的科学观和人生观，踏实奋斗自己的人生。这些都将是我一生中的宝贵财富。

我还要感谢张思东教授。他给予了我无私的指导和帮助，从他身上我学到了很多做人的道理和研究的方法，感谢他在我遇到困难的时候给我的关心和照顾。

感谢和我一起学习、工作的朋友王江林老师、卢小青硕士、赵耀峰硕士、商超硕士、郜帅硕士、张春青硕士、宋育芳硕士、潘冬辉硕士等的支持和帮助。我们在一起共同探讨、共同进步。很怀念同大家一起渡过的时光。

感谢我的家人和朋友，你们始终不渝的关心是我进步的源泉和动力。我的每一个进步，都凝刻着他们无私的支持和关爱。

衷心祝愿我的老师和同学们身体健康、工作顺利、事业有成、万事如意；祝愿我的家人平安幸福、快乐健康。

宗瑞锐

2004 年 2 月于北京交通大学