2017 届研究生硕士学位论文

分类号：_____ 学校代码：10269

密　级：_____ 学　　号：51141500032

華東師範大學

East China Normal University

硕 士 学 位 论 文

MASTER'S DISSERTATION

论文题目：基于 Maude 的 PKMv3 协议形式化建模与验证

院　　系：	计算机科学与软件工程学院
专 业 名 称：	软件工程
研 究 方 向：	高可信计算理论与技术
指 导 教 师：	张民 副教授
学位申请人：	佘 葭

2017 年 4 月

Dissertation for master degree in 2017 University Code: 10269

Student ID: 51141500032

EAST CHINA NORMAL UNIVERSITY

Formal Modeling and Verification of PKMv3 Protocol Using Maude

Department:	School of Computer Science and Software Engineering
Major:	Software Engineering
Research direction:	Trustworthy Computing Theory and Technique
Supervisor:	Assoc Prof. Zhang Min
Candidate:	She Jia

2017.04

华东师范大学学位论文原创性声明

郑重声明：本人呈交的学位论文《基于 Maude 的 PKMv3 协议形式化建模与验证》，是在华东师范大学攻读硕士/博士（请勾选）学位期间，在导师的指导下进行的研究工作及取得的研究成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。

作者签名:_____ 日期: 年 月 日

华东师范大学学位论文著作权使用声明

《基于 Maude 的 PKMv3 协议形式化建模与验证》系本人在华东师范大学攻读学位期间在导师指导下完成的硕士/博士（请勾选）学位论文，本论文的研究成果归华东师范大学所有。本人同意华东师范大学根据相关规定保留和使用此学位论文，并向主管部门和相关机构如国家图书馆、中信所和“知网”送交学位论文的印刷版和电子版；允许学位论文进入华东师范大学图书馆及数据库被查阅、借阅；同意学校将学位论文加入全国博士、硕士学位论文共建单位数据库进行检索，将学位论文的标题和摘要汇编出版，采用影印、缩印或者其它方式合理复制学位论文。

本学位论文属于（请勾选）

（）1. 经华东师范大学相关部门审查核定的“内部”或“涉密”学位论文 *，于年月日解密，解密后适用上述授权。

（）2. 不保密，适用上述授权。

导师签名:_____ 本人签名:_____

年 月 日

* “涉密”学位论文应是已经华东师范大学学位评定委员会办公室或保密委员会审定过的学位论文（需附获批的《华东师范大学研究生申请学位论文“涉密”审批表》方为有效），未经上述部门审定的学位论文均为公开学位论文。此声明栏不填写的，默认为公开学位论文，均适用上述授权）。

余葭 硕士学位论文答辩委员会成员名单

姓名	职称	单位	备注
朱惠彪	教授	华东师范大学	主席
章玥	副教授	华东师范大学	无
毛宏燕	副教授	华东师范大学	无

摘　要

随着科技的发展，人们对无线宽带技术的需求日益增加。全球无线微波载入技术 WiMAX(Worldwide Interoperability for Microwave Acess) 是基于无线城域网络的新技术，实现了宽带接入技术和移动服务的相互结合。IEEE802.16m 标准是在 IEEE802.16e 基础上的补充标准，定义了新一代 WiMAX 技术规范。由于无线系统使用完全开放和未实施保护的无线通信信道，因此需要在无线通信技术中实施可信和强健的安全加密保护实现通信的机密性，隐私性和完整性。

IEEE802.16m 标准在 MAC 安全子层中定义了密钥管理 PKMv3 协议，实现基站与手机端间的双向认证以及安全密钥分发。在 PKMv3 协议中，基站作为服务端，手机站作为客户端，双方通过 X.509 数字证书实现身份验证，建立授权密钥，并通过安全组件协商，实现安全参数交换，最终通过密钥材料的传输，生成传输密钥以加密后续的通讯消息。PKMv3 是改进后的第三代密钥管理协议，目前已有大量研究指出前两代协议中出现较多安全漏洞，并且针对第三代协议的研究仍然不够完善。因此，分析 PKMv3 协议的具体执行流程以及验证协议的安全特性十分关键。

本文对安全协议的研究采用形式化建模的方式。安全协议的形式化方法采用数学模型，实现对协议结构以及通信过程的模拟，并通过有效的程序分析验证系统所满足的性质条件。Maude 是基于重写逻辑的形式化建模语言，它定义了简洁且无二义性的语法，并且提供多种检测方法，适合作为编程语言，算法的分析工具以及系统建模工具。本文通过 Maude 定义的可执行形式化规范，实现对 PKMv3 协议中各执行阶段的建模。不同于现有的研究工作，本文考虑到协议执行过程中密钥的周期性质，在协议模型中加入时间机制模拟密钥重认证的过程，同时引入攻击者模型，模拟入侵者在网络中窃取，重放以及伪造消息的过程。

基于编写完成的 PKMv3 协议可执行规范，本文通过 LTL 模型检测工具实现对协议连续性，密钥活性，认证密钥以及传输密钥生命周期等时间相关性质的检测；并通过穷尽空间状态查找指令实现对协议中机密性，认证性，完整性以及可用性等安全性质的验证。验证结果表明，PKMv3 协议模型能够满足协议标准中的各项时间特性，但可能会遭遇到入侵者攻击，从而无法保证协议的完整性以及可用性。针对协议中的安全漏洞，本文在协议各阶段提出相应的解决方案，进而重新改写协议模型，证明改进后协议所满足的安全特性。

关键词: IEEE802.16m 标准，PKMv3 协议，密钥管理，重写逻辑，Maude 语言，形式化验证

ABSTRACT

With the development of information technology, people's demand for wireless broadband technology is increasing. Worldwide Interoperability for Microwave Access is a new technology based on Wireless Metropolitan Area Network (WLAN), which enables the combination of broadband access technology and mobile services. IEEE802.16m is a supplement standard based on the IEEE802.16e, it defines a new generation of WiMAX technical specifications. Since the wireless system uses completely open and unprotected wireless communication channels, it is necessary to implement trusted and robust encryption protection in wireless communication technology to achieve the confidentiality, privacy and integrity of communications.

IEEE802.16m standard defines the key management PKMv3 protocol in the MAC security sub-layer. It aims to implement mutual authentication and security key distribution between the base station and the subscriber station. In PKMv3 protocol, the base station acts as the server while the subscriber station as the client. They implement mutual authentication by means of X.509 digital certificate to establish the authorization key, and exchange security parameters by means of security association negotiation, and finally generate a traffic key to encrypt subsequent messages. PKMv3 is the third-generation key management protocol. At present, a large number of studies have pointed out security vulnerabilities that exits in the previous two generations of protocols, and the researches on the third generation protocol are still not sufficient. Therefore, it is very important to analyze the specific process of PKMv3 protocol and verify the security property of the protocol.

This paper uses formal modeling method to study the security protocols. The formal

analysis of security protocol adopts the mathematical model to realize the simulation of the protocol structure and the communication process, and then validates the condition that the system can satisfy. Maude is a formal modeling language based on rewriting logic. It defines a simple and unambiguous grammar, and provides a variety of verification methods, which is suitable as a programming language, algorithmic analysis tools and system modeling tools. This paper will implement the modeling of each execution phase of PKMv3 protocol through the executable formalization specification defined by Maude. Different from the existing research work, this paper takes consider of the lifetime of secret keys in PKMv3 protocol, adding time mechanism in the protocol model to simulate the key re-authentication process, and introduces the intruder model to simulate the intruder's behavior of eavesdropping, replaying and faking messages in the network.

Based on the executable specification of PKMv3 protocol, this paper can realize the verification of the time-related properties such as succession, key freshness, period of AK and TEK by means of LTL model checker, and realize the verification of security properties such as confidentiality, authentication, integrity and availability by means of search command. The verification results show that PKMv3 protocol model can meet the time characteristics of the protocol standard. However it may encounter intruder attacks, so it can not guarantee the integrity and availability. In view of the security vulnerabilities in PKMv3 protocol, this paper proposes corresponding solutions in each phase of the protocol, and then adapts the formal specification to prove that the improved protocol can meet safety requirements.

Keywords: *IEEE802.16 Standard, PKMv3 Protocol, Key Management, Rewriting Logic, Maude, Formal Verification*

目录

第一章　绪　论

1.1　研究背景与意义

随着网络时代的发展，当代人类对无线技术的需求愈发增加，不但要求像 3G 技术的移动性和快速切换性，而且希望能得到类似宽带网络一样的高带宽与传输准确性。由此，基于 IEEE802.16m[1] 标准的移动 WiMAX[2] 网络应运而生，WiMAX 是一种城域无线宽带技术，因此被认定为即将成为下一代 4G 技术标准。WiMAX 能够覆盖更大的频率范围以及能够实现更广泛的传输距离，具有更高的可拓展性以及支持 QoS 传输质量保证。由于无线传输媒体十分容易遭受到外部攻击的特性，安全性问题一直在无线网络消息传输中处于至关重要的位置。

在 IEEE802.16m 标准的设计中，一开始便考虑到如何保证无线传输技术的安全性问题。WiMAX 在物理层和 MAC 层利用很多先进技术和系统机制保护信息传输的机密性和完整性，以及抵御不同类型的网络安全攻击。IEEE802.16m 标准在 MAC 层的安全子层中定义了无线宽带连接服务的隐私保护功能，用于实现基站与客户端间建立安全连接。安全子层中主要包括用于加密数据包的封装协议和密钥管理协议 PKMv3 协议 [3]。封装协议包括一系列的密码学套件和 MAC 数据包的转发规则，PKMv3 协议提供了基站与手机端间的身份验证以及密钥分发，实现站点间的密钥材料同步共享。

PKMv3 协议是 WiMAX 技术中的第三代密钥管理协议，在前两代的基础上进行了安全性改进，增加了对消息的分级保护，并在加密运算中采用更加安全的 AES 对称加密算法，并支持 EAP(Extensible Authentication Protocol) 的认证方式和 CMAC(Cipher-based Message Authentication Code) 保护的消息验证。目前针对

PKMv3 协议的研究仍然非常之少，因此分析与检验该密钥管理协议的安全性质十分关键。安全协议的设计目标为实现身份鉴权双方的认证性，关键信息的机密性，传输的完整性，计算资源的可用性以及对目标服务的访问控制。评估并分析安全协议的安全性标准为在该安全协议所处的环境中，协议设计目标中的安全特性是否有可达性缺陷或是否会被人为破坏。

安全协议的形式化方法经过长期的发展，目前已经实现了比较完善的理论体系。协议的形式化分析方法的基本原理是采用逻辑模型，通过有效的程序分析协议系统，判断系统所满足的特定条件以此证明某种协议性质。而传统的人工分析与验证形式化模型，计算效率不高而且会带来因人为疏忽带来的验证结果不准确性。通过自动化协议验证工具，能够很大提高协议分析的工作效率与准确性。

安全协议的形式化分析的理论体系分为三种：基于模态逻辑技术的形式化推理，将协议中的对象及其操作实现逻辑抽象，并通过演绎推理实现主体知识的搜集与推导，从而验证系统是否能够满足某些安全属性；基于模型检测技术的形式化分析方法，将安全协议模拟为分布式系统，主体参与协议过程的行为引发系统局部状态的改变，各局部状态的变化造成整个系统全局状态的改变，检测可达状态所满足的安全属性；定理证明方法，将安全协议定义为定理系统，通过定理证明实现协议安全性质的检测 [4]。

目前形式化方法由于其成本代价小以及高效准确的特性，正逐渐被广泛接受和应用。基于重写逻辑的 Maude[5] 语言是一种高级编程以及形式化建模语言，可以实现分布式以及大型系统的建模。通过 Maude 工具实现对安全协议分析验证，已有很多学者做出相关成功案例。Maude 具有用户友好和直观的形式化语义以及含义，可用于设计与检测系统或者协议模型。Maude 同时支持面向对象以及模块化的规格说明和分析。同时能够提供高效以及稳定的自动化仿真和建模工具，实现系统模型的形式化验证。

相比而言，C、Java 等普通的非形式化语言在描述协议时，对协议通信过程的规格说明太过复杂，可读性较差，对协议性质无法实现清晰的定义，描述语言容易

产生歧义。同时，这些编程语言必须通过线程来模拟系统的并发执行，无法使用高效的规格说明描述分布式系统。并且也没有提供自带的分析工具用于实现对系统状态的穷举搜索，限制了对协议模型的验证能力。

通过 Maude 语言对 PKMv3 协议进行形式化建模，可以实现用户自定义的抽象程度，能够概括描述协议的执行流程，也能够具体定义密钥生成算法。Maude 描述的协议规范，简单易懂，能够准确定义协议性质，并且具有可执行性便于实现协议中各主体消息传输的模拟仿真。通过 Maude 提供的自动化分析工具，能够尽快发现协议设计中的漏洞，从而提高协议设计的安全性 [6]。

1.2 国内外研究现状

由于移动 WiMAX 网络比传统无线网络面临更多的威胁，针对密钥管理 PKM 协议的安全特性已有学者做了大量的研究工作。以下文献通过形式化的分析与建模方法实现对 PKM 三代密钥管理协议安全特性的验证。文献 [7] 利用 BAN 逻辑实现了对前两代密钥管理协议的分析，指出 PKMv1 协议实施单向认证的缺陷以及 PKMv2 协议可能遭遇到的穿插攻击。文献 [8] 使用自动化协议检测工具 Scyther 实现对 PKMv2 协议的形式化分析，验证出协议中机密性、认证性和完整性的缺陷。针对 Scyther 验证得出的协议缺陷，作者在文献 [9] 中提出了一种加入 CA 认证的安全密钥管理协议实现对 PKMv2 的改进。文献 [10] 在 PKMv2 协议的基础上通过加入可信第三方实现站点间的双向认证，并通过 SSM 工具对改进后协议的安全性进行验证。

文献 [11] 使用一种称作 AVISPA 的安全协议自动化按钮工具实现对前两代密钥管理协议的验证，发现协议容易遭受到重放攻击。文献 [12] 通过 Casper 协议建模工具对 PKMv2 协议进行描述，并用 FDR 工具分析进程通信的输出结果，发现入侵者可以截获通信消息进行重放攻击。文献 [13] 同样使用 CasperFDR 工具对 PKMv3 协议中基站、手机站和中继站间传递明文消息的过程进行建模，分析输出结果检测出窃取密钥的攻击方式。文献 [14] 利用 PROMELA 语言对 PKMv3 协议密钥生

命周期的时间特性进行建模，并利用 DT-Spin 工具实现了对协议活跃性，连续性和消息一致性等时间特性的模型检测。

与此同时，也存在不少学者对 WiMAX 网络中存在的安全问题进行理论推导和文献综述，分析与对比密钥管理协议中出现的安全漏洞。文献 [15] 通过文献综述的方式，论述了 WiMAX 中认证和授权的安全问题，如可能遭遇到的 DoS 攻击，存在认证缺陷和密钥空间不足等。文献 [16] 给出基于 IEEE802.16e 标准的移动 WiMAX 网络中安全框架的综述，经调查发现移动 WiMAX 网络易遭受到中间人攻击和 DoS 攻击，因此提出了基于 DH 公钥密码交换的 SINEP 协议以提高网络入口的安全级别。文献 [17] 对 WiMAX 安全子层的结构框架以及密钥管理协议的基本执行原理进行概述，并总结出 WiMAX 能够提供强健的用户认证，访问控制，保护数据隐私性以及完整性，但消息的传输仍然存在阻塞，被窃取以及篡改等重大安全威胁。文献 [18] 描述了 WiMAX 中的安全机制，同时详细列举了各代密钥管理协议中存在的安全缺陷，包括易遭受 DoS 攻击，密钥空间攻击，降级攻击等问题，并且分析总结了目前研究者们提出的相应解决方案进行优缺点对比。文献 [19] 总结了 IEEE802.16e 中密钥管理协议安全协商，双向认证，密钥生成等过程，同时指出为相同安全组件维护不同的传输密钥状态机的时间和空间上的浪费。文献 [20] 对 802.16m 标准中的 PKMv3 协议进行了安全分析，指出协议中存在无法实现快速切换，并且忽略考虑了中继接入问题，以及挑战消息缺乏安全性保护，并且针对各问题提出了相应的解决方案。

与此同时，针对历代密钥管理协议中存在的缺陷，学者们提出不少改进方案。文献 [21] 提出一种改进的安全网络认证协议 ISNAP，在认证消息中采用同时使用随机值和时间戳的方式以防御网络中的重放攻击和 DoS 攻击。文献 [22] 提出尽管 PKMv2 弥补了 PKMv1 协议中的很多安全漏洞，但仍会遭受到重放攻击，并提出与上文类似的随机值与时间戳混合实施保护协议的安全措施。文献 [23] 针对网络中的非法订阅者发送大量请求消息造成基站计算负担的问题，采用可视秘密共享技术以防御 DoS 攻击。文献 [24] 提出了基于消息验证码的共享认证消息技术，预

防网络中的 DDoS 攻击。文献 [25] 探讨了利用标签 ticket 实现目标基站间快速切换的机制，减少了重认证的时间开销。文献 [26] 提出了一种无缝实时的交接算法，描述了 SS 在不同 BS 间的快速以及稳定的切换方式。

随着形式化分析方法的广泛应用，不少学者通过基于重写逻辑的形式化建模语言 Maude 实现了对各协议系统的建模。文献 [27] 描述了基于端到端分布式哈希表的 Kademlia 协议，并通过 Maude 实现对端到端协议系统的抽象与建模，实验模型将不同行为所花费的时间实现参数化，并且通过时间约束的模型检测以及状态搜索工具证明协议各场景下的性质。文献 [28] 通过建立连接 Maude 进程的基础架构，实现通信结点直接向邻居结点或向所有邻居节点组播消息的解决方案，从而在此架构之上实施 EIGRP 协议，允许 Maude 进程中运行的对象能够向更远处发送消息。Olveczky 和 Mesegeur 将 Maude 语言拓展为 Real-Time Maude 工具，从而用于描述和分析实时和混合系统。文献 [29] 提出利用 Real-time maude 形式化模拟，仿真，模型检测实施共享协议的基本技术，并展示了通过该基本技术实例化系统模型，实现对优先级继承协议的验证。文献 [30] 描述利用 Maude 形式化方法实现网络组播协议组成中的 AER/NCA 套件的规格说明和分析验证，Real-Time Maude 被证明十分适用于描述协议系统的成分组合，及其时间敏感性和资源敏感性的行为。文献 [31] 通过 Real-Time Maude 实现无线传感网络的 OGDC 算法的建模，建立的无线网络模型克服描述新的通信形式，地理区域的处理，时间依赖以及概率特性等方面的挑战，同时分析验证了系统的正确性以及处理性能。

本文采用基于重写逻辑的建模语言 Maude，对引入时间机制以及入侵者模型的 PKMv3 协议模型进行形式化分析和验证。Maude 建模语言能够用简洁的语义准确描述网络安全协议的对象属性以及消息传输过程，从而建立可执行的形式化模型，并提供用于 LTL 模型检测和状态空间搜索工具实现对 PKMv3 协议时间以及安全相关特性的验证与改进 [32]。

1.3 研究内容和方法

本文的研究内容为 IEEE802.16m 规范安全子层中定义的 PKMv3 密钥管理协议的执行过程和基本性质。PKMv3 协议实现了 WiMAX 无线网络中基站与手机端间通过 EAP 认证方式的相互认证，从而生成授权密钥建立相互连接，并通过三次握手交换安全组件，实现安全参数协商，进而实施密钥材料传输，最终生成传输密钥用于加密后续通信，实现安全信道消息传输。此外，PKMv3 协议中的授权密钥与传输密钥都具有有限的生命时长，并且传输密钥的生命周期嵌套在授权密钥中。协议主体需要不断执行相应密钥重认证过程，保证双方不间断的通信过程。由此，本文针对 PKMv3 整个生命周期的迭代过程进行时间性相关与安全性相关的详细分析。

本文采用形式化建模的方式实现对 PKMv3 协议的分析与验证。Maude 是基于重写逻辑的形式化规范语言，能够实现对分布式系统的建模与验证。将协议系统中的各部分主体的结构和行为分别建模，最终实现系统全局状态在各主体间信息交互过程中的转换。Maude 对通信协议进行形式化建模与分析的优势在于它能够对协议系统实现准确的定义，其描述的协议代码简单清晰，十分易用。同时，用户能够决定系统的抽象程度，并通过可执行的形式化规范模拟系统运行。并且提供模拟仿真，系统状态检测，LTL 模型检测以及用户自定义的检测工具快速发现协议系统中的漏洞。Maude 同时还能够以更小的代价修改协议系统中的错误，并且通过二次建模验证改进后协议的完备性。

本文提出通过形式化规范语言 Maude 对 PKMv3 协议整个生命周期的建模方法与验证方式。将 PKMv3 协议环境抽象为一个系统，系统中包括发送和接受消息的诚实主体和其他入侵者，以及主体间消息传输与入侵者实施攻击的动态行为。对 PKMv3 协议系统的建模分为以下四个方面：

协议抽象数据类型 通过 Maude 中的函数模块实现协议中各抽象数据类型的定义，包括站点类型，证书类型，消息类型，密钥类型等。并通过操作符定义各数据类型

代表的对象所具有的性质。

协议网络状态组成 通过 Maude 中的函数模块定义协议网络环境的静态组成，包括由主体以及信息集构成的网络节点，网络消息池中的消息组成，以及站点间维护的用于显示密钥生命周期的连接状态。

协议消息传输 通过 Maude 系统模块中的重写规则描述协议各执行阶段消息的传输，实现系统状态的动态转换。详细模拟了 PKMv3 协议中双向认证，安全组件协商，传输密钥生成，加密消息传输以及密钥重认证过程。

入侵者行为 通过 Maude 系统模块中的重写规则描述入侵者对诚实主体间会话实施攻击的行为。定义入侵者在协议执行的不同阶段，实行消息截获，重放旧消息，以及伪造新消息以阻断正常通信过程的具体实施方式。

基于以上建立的可执行协议规范，为其定义初始状态作为执行实例，初始状态中包括具体的站点，传输消息以及连接状态参数。通过 Maude 中自带的验证方式对 PKMv3 协议模型进行以下两个方面性质的验证：

时间特性 通过 Maude 提供的 LTL 模型检测工具实现对协议基本执行过程的连续性，认证密钥以及传输密钥的重认证性等时间相关性质进行验证。

安全特性 通过 Maude 提供的状态空间搜索指令实现对协议中消息传输完整性，计算资源可用性以及密钥机密性等安全相关的性质进行验证。

1.4 论文结构

本文一共分为七个章节：

第一章描述了本文的研究背景，简略介绍密钥管理 PKMv3 协议的安全职能，并引出形式化建模语言 Maude，说明通过 Maude 实现 PKMv3 协议建模的优势。同时介绍相关课题国内外研究现状，以及本文针对 PKMv3 协议的具体研究方法。

第二章对 Maude 语言进行了的详细介绍，包括它的理论基础重写逻辑，Maude 函数模块以及系统模块的语法描述，以及 Maude 所提供的各检测工具的使用方法。同时通过具体的代码示例说明 Maude 规范的编写和验证方式。

第三章针对 IEEE802.16m 规范安全子层中的密钥管理 PKMv3 协议，具体描述其网络层次以及拓扑结构，并详细分析 PKMv3 协议中的双向认证阶段，安全组件协商阶段，传输密钥生成阶段，加密消息传输阶段以及重认证阶段中基站与手机端交换信息的具体执行过程。

第四章通过 Maude 语言实现对 PKMv3 协议各执行阶段的建模。通过 Maude 中的函数模块定义协议中的各抽象数据类型以及系统状态，通过系统模块中的重写规则描述站点间的消息传输，实现系统状态转换。在模型中引入时间机制，实现密钥生命周期的迭代。同时，通过重写规则描述网络中入侵者的攻击行为。

第五章基于已建立的 PKMv3 协议模型，利用 Maude 提供的系统状态搜索指令以及 LTL 模型检测工具对 PKMv3 协议模型中的时间特性以及安全特性进行形式化验证，并且总结与分析本文验证结果与他人工作的异同。

第六章针对 PKMv3 协议中检测出的安全缺陷，提出一种改进后的密钥管理协议，详细定义了协议各执行阶段消息传输的具体内容以及消息保护方式。进而通过 Maude 实现该安全密钥管理协议的形式化分析与验证，得以证明本文提出的改进方案能够成功弥补 PKMv3 协议安全漏洞。

第七章用于总结全文工作，首先总结了本文的研究内容，主要贡献，文章所用的实验方法以及验证结果。其次分析了本文实现方法的不足与局限性，提出替代的建模和验证方式，以及未来可以进行拓展的研究方向。

第二章　Maude 介绍

Maude 是一个基于重写的数学规格说明语言，同时也是一个高效的重写引擎 [33]。Maude 的底层的重写逻辑描述了并发执行的逻辑变化，非常适合用于形式化的表述系统状态和并行计算。因此，Maude 被广泛用来作为计算机系统和编程语言 [34] 形式化建模和验证的逻辑和语义框架。Maude 的需求规格说明是可执行的，高效的可执行力可以实现原型设计和系统调试。基于 Maude 编写出的可执行需求规范，Maude 工具本身为其提供了大量的形式化分析方法，比如模拟仿真，状态空间检索的可达性分析，LTL 模型检测 [35]，以及用户支持用户自定义的执行策略。

Maude 由 Core Maude 和 Full Maude 两部分组成。Core Maude 用 C++ 语言编写，包括 Maude 语言的语法语义，编译原理，函数模块，系统模块，模块等级，预定义模块，模型检测功能等。Full Maude 是 Core Maude 的拓展，支持面向对象的形式化描述，将模型中具有相同性质的元素抽象为类，对象则为类的一个实例，对象间通过预定义消息机制实现交互 [36]。Full Maude 具有反射性，因此可以通过加入新的设计功能进行拓展，其原有的语义和行为都可以被重写。因此，它可以作为构造检测工具或其它语言运行环境的基础公共基础架构 [37]。

Maude 建模语言主要具有以下三个方面的特点：首先是语言的简洁性，Maude 是一种纯函数式语言，语法结构简单；其次是表达能力强，其底层逻辑为重写逻辑，可用于描述软件系统及逻辑推理系统；同时它的计算性能高，在具体实施过程中，运行速度可与其他高效的编程系统相竞争。因此，Maude 工具有着广泛的应用，可用来编写应用程序，或描述为一种可执行的形式化规范，为算法、系统等提供精确的数学模型，从而作为原型模拟系统的行为，也可以用于系统建模，通过对模型性质的规范进行形式化分析与验证。

2.1 重写逻辑

重写逻辑 [38] 是 Maude 的理论基础，它是一种通用的并发处理建模框架。由于重写逻辑是并发变化的逻辑，因此适用于描述系统状态和处理高度非确定性的并发计算。它具有良好的可拓展性，并为很多编程语言和并发模型提供通用的语义框架。值得一提的是，重写逻辑能够很好的支持面向对象的并发计算，因此 Maude 设计中提供了特定语法来实现面向对象模块的描述。由于重写逻辑的计算和逻辑解释就像是一个硬币的正反两面，同样的道理，计算层级上好的语义框架同时也就是逻辑层级上好的逻辑框架，也就是说，其中的其它逻辑可以很好的由元逻辑表示和实施。因此，Maude 中最有趣的应用是元语言的应用，可以用于创造不同逻辑，定理证明，编程语言，计算模型的可执行环境。

重写逻辑的理论基础是代数语义，代数语义将计算机语言定义为抽象数据结构，并为抽象数据结构定义满足计算机语义的公理体系，代数语义中描述抽象数据结构的语法被称作签名，数据类型间的关系由成员等式逻辑中的关系声明定义，而基于签名所声明的签名的公理，由成员关系等是逻辑中的等式定义 [39]。成员关系等式逻辑 [40]（Membership Equational Theory）描述了成员关系等式理论，可以表示为一个二元组（Σ, $E \cup$ A）。其中，Σ 代表签名，指定了包括数据类型，数据类型的从属关系以及操作符的抽象数据结构。E 表示基于签名 Σ 的集合，集合中包括等式，条件等式，以及成员关系的定义，A 代表为等式所满足性质的集合，如结合律，交换律，单位元等。等式及其相应的属性声明了签名中所定义的操作符的具体运算方法和性质。操作符计算的过程是在基于性质 A 的前提下，将匹配项按照等式 E 的内容从左向右推导，直至得出最简结果。

重写逻辑（Rewriting Theory）描述了重写理论，可表示为四元组 $\Re$ = (Σ, $E \cup$ A, ϕ, R)，其中二元组（Σ, $E \cup$ A）继承于成员等式逻辑，ϕ 表示在签名 Σ 中参与操作符运算的参数，R 代表逻辑中重写规则的集合。每个重写规则具有以下形式：

$$l\colon t \to t' \ \text{ if } \ (p_i = q_i) \wedge (u_j := v_j) \wedge (w_k : s_k) \wedge (t_m \to t'_m)$$

其中 l 表示标签，t 和 t' 是通过签名 Σ 中的操作以及全称量词变元所构造的

项，并具有相同的抽象数据类型。重写规则是可基于条件执行的，一共存在四种不同的条件语句，可以是普通的等式，匹配等式，成员关系条件，或是重写条件。从数学上来看，匹配等式被编译器解释为普通等式，在实际编译过程中，将等式中左边的项例如 u_i 与右边的项 u_j 进行匹配，当且仅当两边匹配成功时，该等式成立。在 u_j 中可能存在一些没有在项 t 和其它条件语句中出现的新的变量，当匹配成功后，这些新的变量由 v_j 规范形式中相应的子项实现实例化。我们的工作中并没有使用到成员条件和重写条件，因此这里省略了对它们的具体解释。给定一个项 t_0，若它存在一个子项 t'_0 匹配上述重写规则中的项 t，通过将子项 t'_0 替换为相应的满足条件语句的替代实例, 则可实现将 t_0 重写为一个新的项 t'。

2.2 Maude 形式化建模规范

模块（Module）是 Maude 中最小的编程单元，Maude 中的每一个模块都描述了程序中的一段相对独立的逻辑理论。在 Maude 中，存在两种类型的模块，包括函数模块（Functional Module）以及系统模块（System Module）。函数模块描述成员等式逻辑，而功能模块则描述重写逻辑。当在 Maude 中将某个系统描述为重写逻辑时，重写逻辑的理论基础成员等式逻辑具体指明了系统的“静止”特性，即系统状态集合的数学结构；而重写规则描述了系统的“动态”特性，也就是系统能够执行的所有可能的变换。在 Maude 中，系统的状态被表示为代数诱导结构，可以是一个元组，一组系统组件，或甚至是一个对象。基于诱导公式的表示方法，Maude 可用于描述有限状态系统和无限状态系统。系统状态的转换由重写规则实现形式化定义。给定两个状态 S1 和 S2，设置 T1 和 T2 为相应的表示这两个状态的项。若存在一条重写规则，使得项 T1 执行一次该规则被重写为项 T2，那么可以称之为系统状态 T2 是状态 T1 的一步可达状态。

2.2.1 函数模块

Maude 中的函数模块以关键字`fmod` 开始，`endfm` 结束，具体定义形式为：

`fmod` <*ModuleName*> `is` <*DeclarationAndStatements*> `endfm`

其中 ModuleName 定义了模块名字，`is` 代表模块内容的开始。Declaration And Statements 描述了模块中的具体声明，包括其它模块的引用，数据类型，成员关系，操作，等式的定义。其它模块的引用通过关键字`protecting`，`including` 以及`extending` 实现，从而被引用的模块可作为当前模块的子模块。不同引入关键字的实现方式区别体现在是否将拓展定义以及歧义定义带入子模块 [41]。数据类型的定义通过关键字`sort` 实现，关键字`sorts` 可同时定义多个数据类型，多个数据类型间的从属关系由`subsort` 关键字表示。数据类型的操作运算符用关键字`op` 声明，表述形式为`op` $f : S_1...S_N \rightarrow S$，其中 $S_{i(i=1,..,n)}$ 和 S 代表数据类型。Maude 同时允许用户定义的 mixfix 操作符，并在操作符中使用下划线来指示每个参数的位置。操作分为两种类型，一种是构造函数，另一种代表数值运算，具体操作内容需要通过等式定义。Maude 中的等式表达形式为`eq` $t = t'$，其中 t 和 t' 是类型相同的两个项，等式运算通过从左向右推导，直至无法继续匹配并得出最终结果。等式可以是基于条件的，用关键字`ceq` 声明，并在等式最后附上`if` 关键字定义的条件的集合。

接下来通过哲学家算法为例描述 Maude 中函数模块的编写方法。函数模块中定义了哲学家算法中的基本类型及其相关操作。哲学家算法的背景可以描述为在一张餐桌上围坐着五位哲学家，每位哲学家面前都摆放着一个装满食物的餐盘。餐桌上还摆放着五把叉子，分别放置在每位哲学家左右。餐桌上的哲学家们在初始时刻，都在思考哲学问题，此时哲学家们的状态为 thinking，当其中的某位哲学家感受到饥饿成为 hungry 状态时，需要申请拿起左边以及右边的叉子组成一对后才能开始吃饭，此时哲学家的状态为 eating。当哲学家将餐盘中的食物吃完后，便将左右两把叉子放回原处，以供其他饥饿的哲学家使用。

定义函数模块`PHILOSOPHER` 描述逻辑实体哲学家的相关操作。在该模块中，使用关键字`protecting` 引入 Maude 预定义模块`NAT`，`NAT` 模块中定义了自然数类型`Nat` 的操作。通过`sort` 关键字定义`Phi` 数据类型代表哲学家，以

及Status 类型代表哲学家当前所处的状态。同时，使用关键字op 实现构造函数phi 的定义。phi 操作的输入变量类型为自然数Nat，输出变量类型为哲学家Phi。因此在 Maude 程序中五位哲学家通过编号标识，分别为phi(1)，phi(2)，phi(3)，phi(4) 和phi(5)。关键字ops 定义了通过函数thinking，eating 和hungry 构造出哲学家所处的思考，吃饭和饥饿状态。

```
fmod PHILOSOPHER is
  protecting NAT .
  sort Phi . sort Status .
  op phi : Nat -> Phi .
  ops thinking eating hungry : -> Status [ctor] .
endfm
```

与此同时，定义函数模块FORK 描述逻辑实体叉子的相关操作。首先引入预定义模块NAT，并通过关键字sorts 定义Fork 和Forks 数据类型，分别代表某把叉子和叉子的集合，使用关键字subsorts 将Fork 类型定义为Forks 类型的子集，即指定任意Fork 类型都属于Forks 类型。构造函数empty 声明了某个哲学家手中叉子数为 0 的情况。使用构造函数 fork 将桌上的各叉子通过数字编号区分，则五把叉子可分别表示为fork(1)，fork(2)，fork(3)，fork(4) 和fork(5)。使用 mixfix 操作符__ 实现多个叉子集合的连接，并通过空格符号区分隔开。操作末尾方括号中的内容代表该操作满足的属性，comm 代表交换率（Commutative Law），assoc 代表结合律（Association Law），id: empty 表示单位元（identity）为构造函数empty。

```
fmod FORK is
  protecting NAT .
  sorts Fork Forks . subsort Fork < Forks .
  op empty : -> Forks [ctor] .
  op fork : Nat -> Fork .
  op __ : Forks Forks -> Forks [comm assoc id: empty] .
endfm
```

2.2.2　系统模块

Maude 中的系统模块以关键字mod 开始，endm 结束，具体定义形式为：

mod <*ModuleName*> is <*DeclarationAndStatements*> endm

其中 ModuleName 定义了系统模块的名字，`is` 代表系统模块内容的开始。Declaration And Statements 描述了模块中的所有的定义和声明，包括与函数模块相同的其它模块的引用，数据类型，成员关系，操作，等式的定义，变量的定义，以及无条件的和带条件的重写规则的定义。Maude 中重写规则的语法结构为：

`rl` [<*Label*>] : <$Term1$> => <$Term2$> [*Attributes*] .

`crl` [<*Label*>] : <$Term1$> => <$Term2$> `if` *Conditions* [*Attributes*] .

通过`rl` 关键字定义无条件重写规则，其中`Term1` 和`Term2` 是具有相同数据类型的项，数据项中可能包括多个变量。*Label* 定义了协议的标签，Maude 中允许存在省略标签的规则。一条规则本质上描述了系统的本地并发状态转换：在分布式系统中，只要存在本地实例与`Term1` 匹配，该实例就会被替换为与`Term2` 项匹配的实例。并且如果一个系统的不同实例与多个规则项匹配，那么所有的转换可以同时执行。带条件的重写规则由`crl` 关键字定义，其中 *Conditions* 代表条件等式的并集，在 Maude 中，条件语句按照从左到右的顺序判定，若返回结果值均为 true，再进行重写规则中项的匹配与转换。

本文依然通过哲学家问题为例介绍系统模块的编写方式。系统模块描述了哲学家问题中系统状态的组成以及状态间的转换。首先，通过关键字`protecting` 引入之前定义的`PHILOSOPHER` 和`FORK` 函数模块。其次，定义系统状态为`State` 类型，`State` 类型的系统状态由系统中各哲学家的状态及其拥有的叉子个数，和叉子的总体使用状况组成。各哲学家的状态及其拥有的叉子个数通过构造函数`p[_]:_,_` 表示，该构造函数包括三个输入参数，第一个参数类型为哲学家标识`Phi`，第二个参数类型为该哲学家状态`Status`，第三个参数类型为自然数`Nat`，代表该哲学家当前获取的叉子个数。餐桌上叉子的使用情况由构造函数`forks:_` 表示，其中输入参数为系统当前可用叉子集合。因此，整体系统的状态是各分布式个体的状态的集合，从而使用满足结合律和交换律`__` 操作连接个体状态`State`。系统的初始状态由操作符`init` 表示，并通过`eq` 关键字定义具体数值。在程序的初始状态，存在五位陷入思考状态的哲学家，每位哲学家手中的叉子数量均为 0，从而餐桌上的

五把叉子都处于空闲状态，可被随时申请。

通过四条重写规则描述系统中各哲学家的行为。标签为hungry 的重写规则描述了当感到饥饿时，哲学家的状态由thinking 转变为hungry，其它参数保持不变，其中N 和F 是自然数类型的变量，N 代表任意某个哲学家的标号，F 代表其拥有任意叉子的数量。带条件的getfork 规则描述了当哲学家处于饥饿的状态，若他的左边或右边存在空闲的叉子时，他可以一次取到一个叉子。此时，该哲学家所拥有的叉子数量加一，餐桌上可用叉子的数量减少，变量NS 表示餐桌上N1 以外其它叉子的集合。If 条件语句中的canUse 操作定义了每个哲学家可以申请与之编号相同的叉子，以及编号小一位的叉子，但如果是编号为 1 的哲学家，他仅可以取得 1 号和 5 号叉子。eat 规则描述了当处于饥饿状态的哲学家获取到两把叉子后，他可以开始此享用食物，他的当前状态由hungry 转变为eating。putfork 规则描述了当哲学家吃完后将两只叉子放回原处的过程，此时哲学家的状态由eating 转变为thinking。if_then_else 语句描述了若是第一位哲学家，则将第二个叉子放到编号为 5 的位置上，若是其他哲学家，则放在编号比自身减一的位置上。由此，实现了对各位哲学家所有可能行为的分布式系统的描述。

```
mod DINING is
  protecting PHILOSOPHER . protecting FORK .
  sort State .
  op p[_]:_,_ : Phi Status Nat -> State .
  op forks:_ : Forks -> State .
  op __ : State State -> State [assoc comm] .
  op canUse : Nat Nat -> Bool .
  eq canUse(N,N1) = (N == N1) or (N == N1 + 1)
  or (N == 1 and N1 == 5) .
  op init : -> State .
  eq init = ((p[phi(1)]: thinking, 0) (p[phi(2)]: thinking, 0)
          (p[phi(3)]: thinking, 0) (p[phi(4)]:thinking, 0)
          (p[phi(5)]: thinking, 0))
          (forks: (fork(1) fork(2) fork(3) fork(4) fork(5))) .
  vars N N1 F : Nat . var FS : Forks .
  rl [hungry] : (p[phi(N)]: thinking, F) => (p[phi(N)]: hungry, F) .
  crl [getFork] : (p[phi(N)]: hungry, F) (forks: (fork(N1) FS)) =>
     (p[phi(N)]: hungry, F + 1) (forks: FS) if canUse(N,N1) .
  rl [eat] : (p[phi(N)]: hungry, 2) => (p[phi(N)]: eating, 2) .
  rl [putFork] : (p[phi(N)]: eating, 2) (forks: FS) =>
            (p[phi(N)]: thinking, 0) (forks: (FS fork(N)
            fork(if N == 1 then 5 else sd(N,1) fi) )) .
endm
```

2.3 Maude 形式化分析方法

Maude 中的交互命令`reduce`（简写为`red`）实现了表达式的计算。Maude 将不断重复执行该`red` 命令用于将等式操作中等号左边的内容替换为等式右边的内容，并打印出最终的执行数据，打印出的数据为最简结果 [42]。以之前定义的哲学家算法为例描述`red` 指令的语法。通过`load` 关键字载入相关函数模块以及系统模块，在 Maude 命令行中输出以下`red` 指令。哲学家算法中定义了操作`canUse` 的等式运算，当哲学家编号为 1 时，只能请求桌上编号为 1 或为 5 的叉子，因此执行结果返回为 false，Maude 同时打印出执行该命令的重写次数与时间。

```
Maude> red canUse(1,4) .
reduce in DINING : canUse(1, 4) .
rewrites: 15 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: false
```

Maude 原型设计的模拟仿真通过`rewrite` 重写指令（简写为`rew`）实现。它在一个给定的数据项上不断执行定义在 Maude 系统模块中的重写规则，使得该数据项逐步实现转换，数据项的一步步转换过程模拟了系统的某个指定行为。当使用重写命令执行重写规则时，若有些规则可以应用无限次，在这种情况下，重写步骤的上限需要强制 Maude 停止。同时 Maude 也提供公平重写指令`frewrite`（简写为`frew`），该指令采用深度优先的位置公平策略，使得处于偏远位置的重写规则也能够得到执行。在 Maude 中重写命令的语法如下所示：

`rew` [[n]] [`in` $module$:] $initState$.

`frew` [[n]] [`in` $module$:] $initState$.

其中，方框中的内容 n 以及 $inmodule$ 都是可选参数，n 代表执行重写规则次数的上限，$inmodule$ 代表只匹配该系统模块中的重写规则。$initState$ 表示给定的初始项集，若不填写可选内容，则 Maude 从初始项集开始，不断执行系统中匹配的重写规则，直至最终状态。以哲学家算法为例，执行`rew` 指令将系统中只有编号为 1 的哲学家申请到一把叉子的情况作为初始项集，查看其匹配一条重写规则得到的结果。返回的结果显示编号为 1 的哲学家集齐两只叉子。

```
Maude> rew [1] in DINING : ((p[phi(1)]: hungry, 1) (p[phi(2)]: thinking, 0) (p[
    phi(3)]: thinking, 0) (p[phi(4)]: thinking, 0) (p[phi(5)]: thinking, 0)) (
    forks: (fork(2) fork(3) fork(4) fork(5))) .
rewrites: 62 in 0ms cpu (0ms real) (~ rewrites/second)
result State: forks: (fork(2) fork(3) fork(4)) (p[phi(1)]: hungry,2) (p[phi(2)]:
    thinking,0) (p[phi(3)]: thinking,0) (p[phi(4)]: thinking,0) (p[phi(5)]:
    thinking,0)
```

由于重写指令rew 和frew 只能从初始状态探索到系统一个可能的行为（通过执行一系列重写规则），因此，Maude 提供search 指令探索给定系统所有可能的状态空间，search 指令可以从初始状态开始执行不同的重写路径，从而模拟不同的行为状态。search 指令还可以用于检测系统的不变性质（Invariant Property），系统的某个不变性质是指坏的事情永远不会发生。将该不变性质的对立面通过search 指令中的条件语句描述出来，检测执行结果是否存在解决方案。若执行结果返回值不为空，则这个解决方案为不变性质的反例。如果系统状态的个数是无限的，或者系统状态的个数超出了有限的时间和内存，可为search 指令设置一个最大搜索深度，在这种情况下不变性质只能被称作是部分满足的。Maude 中search 指令的语法如下所示：

search [n,m] [in $Module$:] <$Term1$> =>! <$Term2$> such that $Condition$.

其中 [n,m] 是可选参数，n 代表所需解决方案的上限，m 表明搜索状态空间的最大深度。$Module$ 代表执行重写逻辑的模块，若省略则表示执行整个代码范围内的重写逻辑。Term1 给定了初始项集，Term2 表示从初始项集可达的模式匹配。$Condition$ 代表可选的 if 条件语句，这里的 if 条件语句与带条件的重写规则中具有相同的形式，if 条件语句声明了可达状态需满足的不变性质。隐藏的中间项T 是在满足可选条件的情况下T 与Term2 匹配，并且存在从Term1 到T 的重写序列的解。箭头后的符号用于规定序列中的重写步骤的长度，符号! 意味着T 必须是不能继续重写的项集，代表可能的最终状态，亦可用1，+ 和* 符号替换，分别表示执行依次一次重写，一次或多次步骤，零或一次或多次重写。

通过以下search 指令查找从哲学家算法程序中定义的初始状态开始，是否存在不可重写的最终状态，即哲学家算法的死锁状态。返回结果得出一条执行路

径，显示当五位哲学家各申请到一把叉子时，程序出现死锁，在这种状态下，没有哲学家可以继续申请到两把叉子，无法匹配任何重写规则。

```
Maude> search init =>! S:State .
search in DINING : init =>! S:State .
Solution 1 (state 1964)
states: 2110 rewrites: 207041
S:State --> forks: empty (p[phi(1)]: hungry,1) (p[phi(2)]: hungry,1) (p[phi(3)]:
    hungry,1) (p[phi(4)]: hungry,1) (p[phi(5)]: hungry,1)
No more solutions.
states: 2163 rewrites: 211565
```

对于一个给定的系统模块，其中所描述的规格说明可以分为两个层次，一个是系统规范层次，由描述系统行为的重写规则定义；另一个是性质规范，由申明和证明模块特性的某些性质定义。Maude 工具中可以实施有效的 LTL 模型检测从而验证系统的线性时态逻辑性质（Linear Temporal Logic）。给定一个系统的 Maude 规范，需要先在系统状态上定义一组原子命题，实现系统的性质规范。时序逻辑允许安全属性（Safety Property）和活性属性（Liveness Property）的性质规范，安全属性是指坏的事情永远不会发生，活性属性是指好的事情永远都会发生。Maude 的 LTL 模型检查器基于初始状态和表述原子命题逻辑关系的 LTL 公式，返回模型检测结果。在 Maude 中进行 LTL 模型检查的条件，是从给定初始状态到可达到的状态集合必须是有限的。

Maude 预定义`LTL-SIMPLIFIER` 模块中定义 $\models$ 操作，用于实现对原子命题的描述。原子命题 (Proposition) 的数据类型为`Prop`，满足该原子命题的系统状态为`State` 类型，若符号左边的系统状态符合右边的原子命题，则定义结果为 true。预定义`MODEL-CHECKER` 模块中定义了`modelCheck` 操作，该操作以某个特定的系统状态和 LTL 公式为输入参数，并得出模型检测结果。若定义的模型满足 LTL 公式描述的系统性质，结果返回布尔值 true，若不满足性质则返回一条反例。

```
op _|=_ : State Prop -> Bool [frozen] .
op modelCheck : State Formula ~> ModelCheckResult [special(...)] .
```

以下代码通过 LTL 模型检测进行验证，检测哲学家问题中编号为 1 的哲学家吃饱的状态是否始终能达到编号为 2 的哲学家吃饱的状态。首先引入 model-checker 文件中`MODEL-CHECKER` 模块和`LTL-SIMPLIFIER` 模块。其次定义两个原子命

题fullup1 和fullup2 分别代表编号为 1 和编号为 2 的哲学家吃饱的状态。LTL 公式[]fullup1 → <>fullup2 描述了每个哲学家 1 吃饱的状态最终总能达到哲学家 2 吃饱的状态，其中符号[] 代表 Henceforth，<> 代表 Eventually，-> 代表 Implication。执行red modelCheck 语句，返回的布尔值结果为 true，证明哲学家模型满足这一性质。

```
mod MODEL-PHI is
 protecting DINING .
 including MODEL-CHECKER .
 including LTL-SIMPLIFIER .
 var S1 : State .
 ops fillup1 fillup2 : -> Prop .
 op ((p[phi(1)]: eating,2) S1) |= fullup1 = true .
 op ((p[phi(2)]: eating,2) S1) |= fullup2 = true .
endm
Maude> load model-checker.maude
Maude> red modelCheck(init, [] fullup1 -> <> fullup2) .
reduce in MODELS : modelCheck(init, []fullup1 -> <> fullup2) .
rewrites: 19 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

2.4　本章小结

本章介绍了形式化建模语言 Maude 的理论基础为重写逻辑，基于成员等式逻辑以及重写逻辑可实现 Maude 中函数模块以及系统模块的定义。并以哲学家问题为例，描述通过函数模块定义数据结构及其性质的具体过程，以及通过系统模块模拟哲学家的行为的方法。此外，阐述了 Maude 工具中提供的各验证方法，如 reduce、rewrite、search 指令以及 LTL 模型检测工具的使用场景以及具体使用方式。

第三章　PKMv3 密钥管理协议

3.1　IEEE802.16m 标准

IEEE802.16 被称作局域网/城域网标准委员会 LMSC(LAN /MAN Standards Committee)，致力于研究 OSI 参考模型中的物理层和及介质访问层规范 [43]。IEEE802.16 是宽带无线技术 WiMAX(Broadband Wireless MAN Standard) 的标准，无线宽带接入 BWA(Broadband Wireless Access) 与无线局域网络标准 802.11[44] 和短程无线通信标准 802.15[45] 互补，填补了无线接入标准的空白。IEEE802.16 为用户站点和核心网络提供无线通信路径，这种无线访问标准解决了城域网中的最后一公里问题。

这一系列的 IEEE802.16 标准中可根据是否支持移动性的特性被区分为移动宽带无线接入空中接口标准和固定宽带无线接入空中接口标准两种类别，其中 802.16a、80216d 属于固定宽带无线接入空中接口标准，而 802.16e 和 802.16m 等则属于移动宽带无线接入空中接口标准 [46]。未来通信领域的发展趋势表现出，从传统宽带固定接入用户的角度：仅仅在家庭、办公室等固定领域使用宽带业务已无法满足人们的需求，希望宽带服务能接入移动服务，从传统移动用户的角度：短认和低速的数据业务也已不满足需求，希望能够使用更高速率的业务，从而宽带接入服务和移动技术应逐渐实现相互融合 [47]。

3.1.1　IEEE802.16m 标准的特点

IEEE802.16m(802.16-2011) 是 IEEE802.16 系列标准的新一代革命性更新，也是继 802.16e 之后的新一代 WiMAX 标准。因此 IEEE802.16m 被称认定为准 4G 标准，又称作 Mobile WiMAX Release2。从严格意义上讲，802.16m 不只是一个移动通

信系统的标准，它同时还是一个无线城域网的技术。它的设计目标是向终端用户提供 100Mbps 的下行速率，其中包括多输入输出技术 MIMO(Multiple Input Multiple Output) 以及质量检测技术 QoS(Quality of Service)，它同时还支持用于小型家庭基站和自组织网络。

相较于 3G 技术来说，WiMAX 采用 OFDM/OFDMA、AAS、MIMO 等先进技术，因此它的技术起点相对要求较高。随着技术标准的不断发展，WiMAX 致力于实现移动化的宽带业务，而标准 3G 则将实现宽带化的移动业务。从标准来说，在高速移动的情况下 WiMAX 很难达到在移动过程中做到无缝切换的要求，其性能与主流 3G 技术相比还有一定的差距。与 WiFi 技术相比较，WiMAX 技术具有大频率的覆盖范围，对固定站点支持长达 50km 的传输范围，对移动站点则支持长达 15km 的传输范围，有效弥补了 WiFi 信号覆盖面积较小的不足。WiMAX2 网络和天线技术复杂，因此各种环境下都可以获得最佳性能，但与 WiFi 技术相比其所需的功耗较大。WiMAX 具有可扩展性，支持灵活的信道带宽和信道复用，扩展能力较强。与之相比 WIFI 技术的扩展能力就较差，当用户数量多时，信道容易产生碰撞。此外 WiMAX 还提供了 WIFI 技术无法支持的 QoS 保障。

3.1.2 IEEE802.16m 网络拓扑结构

在 WiMAX 的空中接入网络 AAI(Advanced Air Internet) 中，存在两种类型的站点，包括用于提供服务的中央无线基站 BS(Base Station) 和请求服务的可移动设备 SS(Subscriber Station)。IEEE802.16m 支持两种拓扑方式，分别为点到多点的 PMP 结构以及网状结构 Mesh。在 PMP 网络拓扑中，基站 BS 作为服务端中心，与多个手机站 SS 连接并提供认证服务，而在 Mesh 模式下，网络中的手机站 SS 可以与基站和另外的手机站 SS 相互通信，可根据实际情况灵活的部署，实现无线设备间的消息传输和网络的弹性延伸 [48]。

网络参考模型 NRM(Network Reference Model) 是 IEEE802.16m 网络架构的逻辑表示，具体结构如图 3.1 所示，由功能实体以及相互交互的参考点所组成。

IEEE802.16m 网络参考模型包括接入网 ASN(Acess Service Network) 和连接服务网 CSN(Connection Service Network)。接入网 ASN 中的功能实体包括各个提供认证服务的基站 BS 以及连接各 BS 提供无线接入服务的接入网网关 ASN GW(ASN Gateaway)。连接服务网 CSN 为 ASN 提供 IP 连接服务，其中包括用于认证的 AAA 服务器，IP 路由，FTP 文件传输，防火墙等。

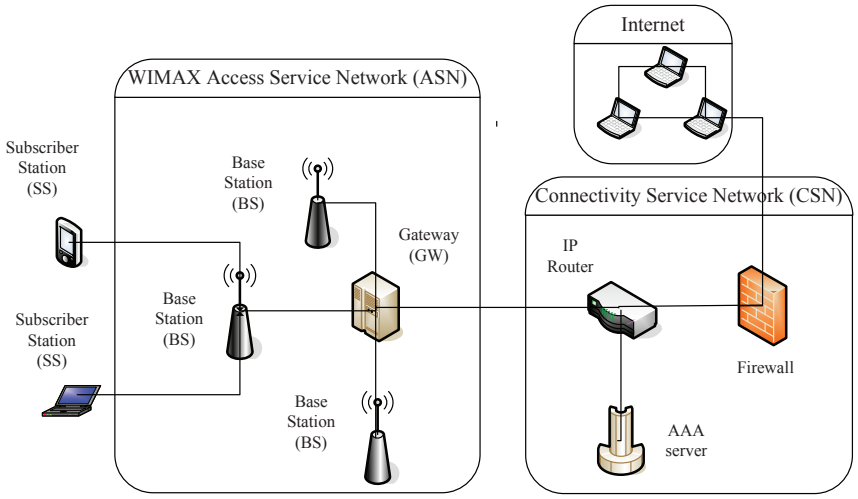

图 3.1: IEEE802.16m 网络拓扑图

每一个 ASN 网关以及其连接的多个基站 BS 构成了一个寻呼组（Paging Group），在接入网 ASN 中，存在多个这样的寻呼组。基站处于各寻呼组的中间位置并为各手机站提供无线连接服务，ASN 网关处于寻呼组边界位置，它实现了多个基站流量的控制与集聚，使之与连接服务网相连接，并且管理了不同基站间的安全交接工作，其中包括维护认证过程，传输服务流以及密钥分发。连接服务网是整个网络的核心，它通过以上提到的一系列服务，如 AAA，FTP，DHCP 等管理基站和接入网网关，也提供了到达其它网络的交互操作和移动漫游技术。

3.2　IEEE802.16 参考模型层次结构

IEEE802.16m 规范的协议栈模型如图 3.2 所示，从纵向上定义了 OSI 参考模型的物理层，介质访问层以及层次间的接口。并从横向上定义了模型的数据/控制平面和管理平面。数据处理过程中各种具体功能的处理转发通过数据平面来实现，网

络协议的运行受控制平面的控制和管理，为数据平面提供数据处理所必须的各种网络信息，管理平面由网络管理人员使用，它被用来支持、理解和执行网络相关管理人员对于相关设备的网络协议设置命令。

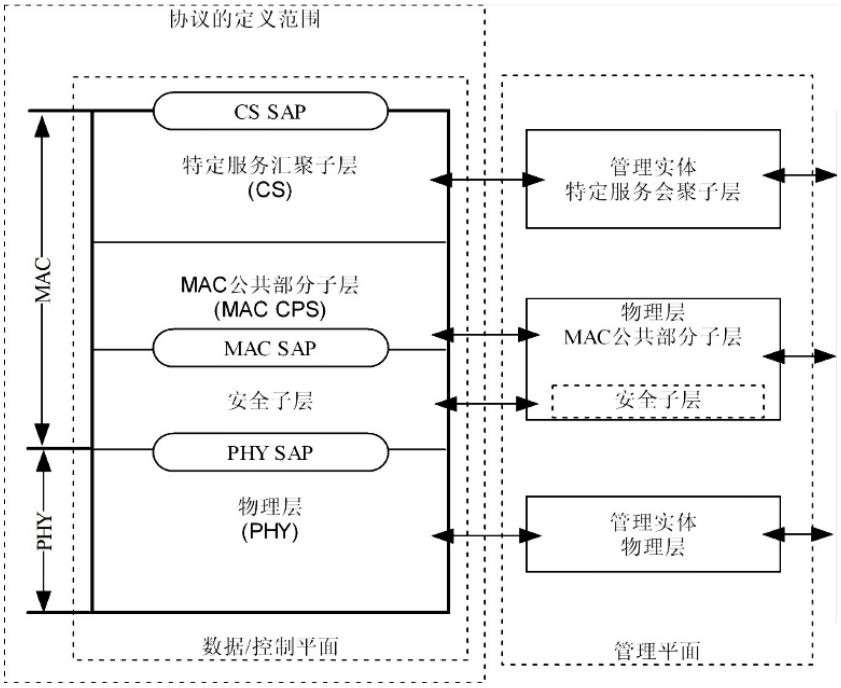

图 3.2: IEEE802.16m 网络参考模型

3.2.1 物理层

IEEE802.16m 物理层定义了 10 到 60GHz 的操作频段，实现了对 MAC 层 PDU 的汇聚、编码、调制，最后形成无线帧送入物理信道 [49]。它被设计成为高度灵活系统，从而为服务提供者，包括基站部署，无线接入服务等实现最优的系统部署。IEEE 802.16 的物理层支持单载波和多载波的传输，实现了信息的单播和组播，并同时支持四种类型的物理层规范，其中两种基于单载波的物理层规范，分别为 WirelessMAN-SC 和 WirelessMAN-SCa，SC 的操作频段为 10 66 GHz，且为视距 (LOS) 操作；而 SCa 的操作频段低于 11GHz，为非视距 (NLOS) 操作；另两种基于多载波的物理层规范：WirelessMAN-OFDM 和 WirelessMAN-OFDMA，这两种规范的操作频段均低于 11GHz，并都基于 OFDM 多载波技术 [50]。

3.2.2　介质访问控制层

介质访问 MAC 层处于协议栈中物理层上方，它在结构上被分为三个子层，包括特定服务汇聚子层 CS(Convergence Sub-layer)：转换或映射来自外部网络的数据，定义多种 CS 层规范并提供与外部网络的接口；MAC 公共部分子层 CPS(Common Part Sub-layer)：提供 MAC 层的核心功能，比如系统接入、带宽分配、连接建立以及连接维护等服务；MAC 安全子层（Security Sub-layer）：实现认证授权，密钥分发，访问控制，数据包加密等功能，定义和维护 WiMAX 的安全特性 [51]。MAC 服务接入点 SAP(Service Access Point) 处于在 MAC 层各子层间，并作为接口实现各层级间的信息传输和通信。

特定服务汇聚子层实现高层协议数据单元的接收，并将接收到的 PDU 映射到 MAC 层 [52]。它的主要功能包括为对接收到的高层 PDU 进行分类操作，根据分类来对 PDU 进行相应处理，从而为正确的 MAC 服务接入点传递 CS PDU，或从对等层接收 CS PDU。MAC 公共部分子层实现 MAC 层的大部分核心功能，包括 MAC PDU 构建和传送的定义，进入网络和初始化网络，请求带宽和授予带宽，实现竞争解决算法，管理服务流和实现各种链路自适应技术等 [53]。安全子层通过对 BS 和 SS 间传输的 MAC PDU 进行加密和解密操作，为经过 WiMAX 网络的用户数据提供机密性，也被用于为 BS 和 SS 提供认证授权及密钥交换。安全子层支持使用数字证书的身份识别，并提供基于 EAP 的认证方式，从而实现授权保护。通过将授权后 BS 与 SS 间传输的数据进行加密，实现了对机密信息的访问控制。

安全子层的隐私保护功能主要由两个协议组成：用于加密数据包的封装协议（Encapsulation Protocol）和密钥管理协议 PKMv3（Privacy and Key Management Protocol）。封装协议包括一系列的密码学套件和将这些加密算法应用于 MAC 数据包的规则，密钥管理 PKMv3 协议提供了基站与手机端间的密钥分发，实现站点间的密钥材料同步共享。同时，PKMv3 协议实现基站对网络服务的访问控制。在 IEEE802.16m 标准中，用户数据的加密是在 MAC PDU 生成之后进行的，IEEE802.16m 安全功能的另一个特点体现在通过 MAC 保护消息实现了无线网络

中消息传输的完整性保护。

IEEE802.16m 安全架构的功能模块组成如图 3.3 所示。对于 AAI 网络中的 SS 和 BS 而言，安全功能模块可以分为两个逻辑类别：安全管理实体，包括整体安全管理，提供认证框架的可拓展认证协议 EAP 的封装/解封装，随着密钥生成、计算、分发过程的 PKM 控制功能和密钥状态管理，认证和安全组件控制功能以及位置隐私性保护；加密和完整性保护实体，包括用户数据加密和认证，管理消息认证，以及管理消息的机密性保护。值得一提的是，可拓展协议的具体实施过程并不包括在 IEEE802.16m 规范中。

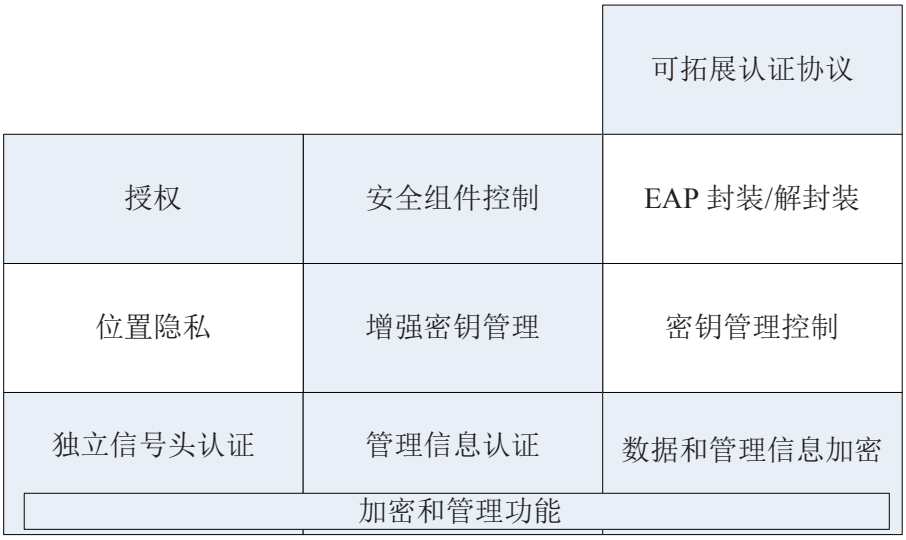

图 3.3: IEEE802.16m 安全架构功能模块

3.3 PKMv3 协议密钥协商流程

密钥管理 PKMv3 协议定义于 IEEE802.16m 安全子层，它提供了从基站到终端的密钥数据的安全分发机制。通过该协议的执行，基站与终端间实现了双向认证与授权，安全组件的协商以及密钥交换。在 IEEE802.16m 的 AAI 网络中，BS 作为认证端，同时 SS 被描述为终端。AAI 网络的安全功能模块通过提供 BS 与 SS 间协议数据单元的加密传输，为 SS 保障了隐私性，认证性和机密性。

PKMv3 协议在前两代密钥管理协议的基础上进行了较大的改进，三代密钥管理协议的区别如表 3.1 所示。PKMv1 协议假定网络提供者是可信的，因此只需要

单方向用户向基站认证，就能够建立双方连接，这很容易造成攻击者伪造为基站获取用户信息的漏洞，也可能会遭受到重写攻击和拒绝服务攻击。PKMv2 协议弥补了前一代协议最大的安全漏洞，支持 RSA 和 EAP 两种认证方式实现 SS 与 BS 间的双向验证，但该版本的协议仍然存在无法支持组播，加密算法 DES 不够安全等协议漏洞。PKMv3 协议只支持基于 EAP 的认证方式以及 CMAC 消息验证方式，能够与 AAA 体系结构灵活交互，支持 AES 加密算法，同时也增加了对消息管理的选择性保护策略 [54]。

表 3.1: 密钥管理协议特性比较

密钥管理协议		**PKMv1**	**PKMv2**	**PKMv3**
特性	认证主体	单向认证	双向认证	双向认证
	认证方法	RSA 认证	RSA/EAP 认证	EAP 认证
	消息保护	无	部分消息保护	消息分级保护
	加密算法	DES 加密	DES 加密	AES 加密
	消息认证码	HMAC/CMAC	HMAC/CMAC	CMAC
	支持组播	不支持	不支持	支持

PKMv3 采用的消息分级保护策略，保护等级可分为如图 3.4 所示的三个层次：

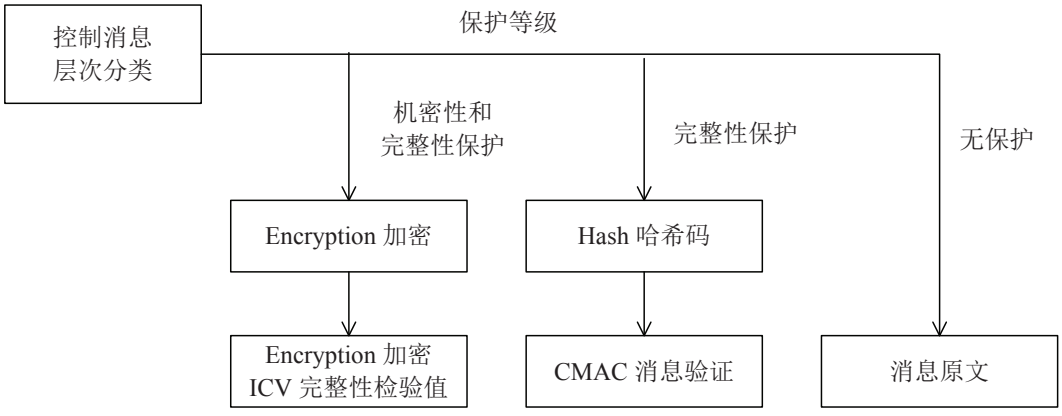

图 3.4: PKMv3 消息保护等级

等级一：基于 AES-CCM[55] 的认证加密和基于 ICV 的完整性保护。CCM 是对称加密算法 AES 的操作模式，提供鉴别加密服务，保证传输准确性，同步性和并行性。完整性检验值 ICV 用于检测在消息传输中消息的完整性是否遭到破坏。

等级二：基于 CMAC 的完整性保护。消息验证码 CMAC 通过将 CMAC 元组加入到 MAC 控制消息中实现整条控制消息的完整性和认证性保护。在实际传输中，受保护的控制消息是通过明文传输的，从而减少了解密操作带来的计算负担。

等级三：消息无保护。如果 SS 和 BS 间没有共享的安全信息或者消息保护是不必要的，那么控制消息则不会被加密或认证。在消息无保护模式下双方不需要协商安全组件。认证阶段之前的控制消息也属于这一类别。

PKMv3 协议的主要目的是实现 SS 与 BS 间的双向认证从而生成传输密钥 TEK 用于加密后续的信息交流。TEK 的衍生过程需要经过四个阶段，包括初始化过程，SS 向 BS 发送自己的 X.509 证书预示整个 PKMv3 过程的开始，此时 BS 并不需要回应；双向认证过程，SS 和 BS 实现相互身份认证并生成认证密钥 AK；安全组件交换过程，双方协商加密算法和其他安全参数；密钥传输过程，SS 和 BS 最终得以生成传输密钥 TEK，并开始安全通信。

3.3.1 双向认证过程

PKMv3 协议支持基于 EAP 认证的方式。EAP 认证的工作流程可以简单归纳为: 首先 SS 向 BS 发送一个或多个请求报文, 其次, BS 对接收到的每一个请求报文回应一个应答报文，最后，SS 通过发送成功或者失败的报文来结束整个认证过程。通过成功的 EAP 认证过程，SS 和 BS 实现了双向认证，以及主密钥 PMK(Pairwise Master Key) 的传输，PMK 用于生成授权密钥 AK(Authorization Key)。AK 可用于演化出后续所需的其他密钥材料，这种分层机制允许在不增加额外计算复杂度的同时，加密密钥的频繁刷新。

在使用 EAP 协议的认证方式中，请求方需通过运营商发行的证书实现身份验证，比如用户身份识别 SIM(Subscriber Identity Module) 卡或者 X.509 数字证书。数字证书由 CA(Certificate Authority) 认证中心发放，其中包括使用者的标识号，公钥信息以及 MAC 地址等。当发起认证请求时，SS 向 BS 发送它自身的数字证书得以验证身份的合法性。每个通过数字证书认证身份的用户在出厂时便带有其公私密

钥对以及 X.509 数字证书。EAP 协议是 IEEE 802.16 认证机制的核心，它可以提供不同的方法分别支持 PPP，以太网、无线局域网的链路验证，从而实现了认证的扩展性及灵活性。目前主流的 EAP 认证方式有基于 SIM 验证的 EAP-AKA(Authentication and Key Agreement)，基于 X.509 证书验证的 EAP-TLS(Transport Level Security)，基于微软挑战握手认证协议的 EAP-TTLS(Tunneled Transport Layer Security) 等。

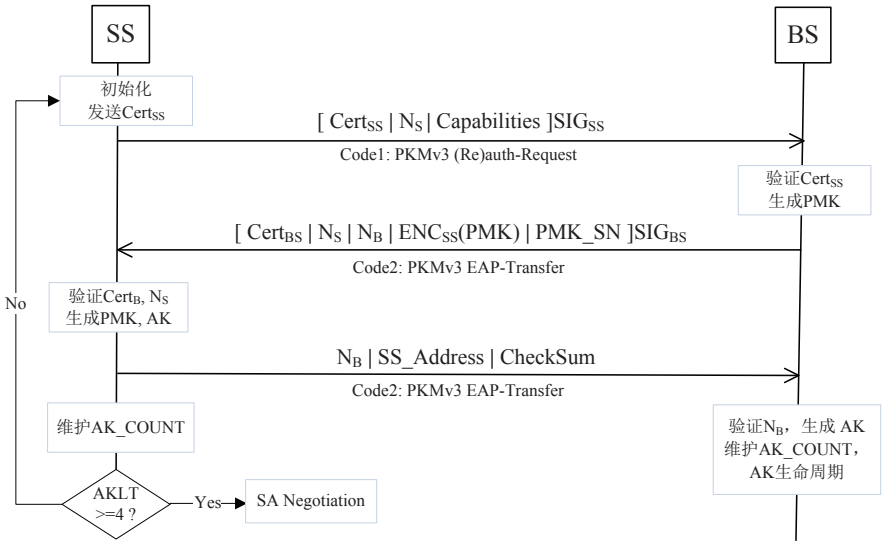

图 3.5: 双向认证通信过程

PKMv3 协议双向认证的通信过程如图 3.5 所示。首先，SS 向 BS 发送认证请求消息 Auth-Request 作为 EAP 会话的开始，发送的消息中包括 SS 的 X.509 数字证书，SS 生成的随机值 N_S 以及其支持的认证和加密算法，并将该消息用 SS 的私钥加密实现数字签名以抵抗攻击者的篡改。其中，随机值用于帮助网络中的通信主体识别连续的请求，从而辨别出攻击者的重放消息并抵抗 DoS 攻击。BS 将 SS 发送的认证请求消息用 SS 的公钥进行解密，若解密消息得到的数字证书的主体与 SS 一致，则 BS 向 SS 发送认证应答消息 EAP-Transfer 作为回应，并用 BS 的私钥对消息进行数字签名作为防篡改保护。消息包括 BS 的 X.509 证书，接收到的 SS 的随机数 N_S，BS 生成的随机数 N_B，用 SS 公钥加密保护的主密钥 PMK，PMK 的序列号。PMK 的序列号长度为 4bits。同样，SS 对接收到的 BS 发送的消息用 BS 的公钥进行解密，认证 BS 数字证书的合法性，以及前两条消息中随机数 N_S 的值

是否一致。

至此，SS 和 BS 间完成相互认证，SS 向 BS 发送认证成功消息作为整个 EAP 传输阶段的结束。消息中包含接收到的 BS 生成的随机数 N_B 以及 SS 地址信息，并附上校验和以确保该消息的完整性。SS 的地址用于与 PMK，BS 的标识号一起生成认证密钥 AK。认证过程完成后，SS 和 BS 通过安全信息的交换均生成 AK，同时还要为每个 AK 维护一个称为 AK_COUNT 的同步计数器，该计数器在 SS 由空闲模式重新进入网络，安全位置更新等情况下递增，从而确保不同的 CMAC key 和 TEK 用于 BS 与 SS 会话中，以保障消息的安全性。CMAC-TEK prekey 由双方利用 AK 和 AK_COUNT 生成，从而计算出用于生成 CMAC 消息摘要的 CMAC key。

3.3.2 安全组件交换

安全关联 SA(Security Association) 是 BS 与 SS 在 AAI 网络中共享的安全信息集合，用于确定之后的消息传输的加密方式以及其他所需的安全参数，是通信双方共同签署的协定。双方所协商成功的安全关联 SA 通过 SAID (SA Identifier) 来标识，它是两个应用实体间的单向逻辑连接，协商出数据保护对象范围，数据保护方式，和数据保护实施者。在双方共同协商出 SA 后，相应的 SAID 所对应的保护级别将应用于各单播数据流（Unicast Flow）中。

在 IEEE802.16m 中，存在三种类型的安全组件 SA，分别是初始 SA（Primary SA），静态 SA（Static SA）和动态 SA（Dynamic SA）。Primary SA 存在于 SS 中，是每一个 SS 初始化程序时建立的一个初始 SA，初始 SA 的值与 Basic CID（Connection Identifier）的值相等；Static SA 存在与 BS 中，AAI 网络的单播传输仅支持静态 SA；Dynamic SA 的建立和删除由通讯中的动态服务流决定。控制消息的分级保护策略的选择方式是由扩展消息头部 AGMH 中的数据流标识符 FID（Flow Identifier）决定的。因此，一条控制消息的加密与否取决于与消息相关的安全组件的等级，也就是初始 SA 的值。

当 SS 和 BS 成功实施双向认证后，双方通过三次握手阶段实现安全组件的协

商，从而确定了适用于后续阶段消息传输的 SAID。存在三种类型的 SAID，与分别表示对数据不同的保护等级其中，SAID 0x00 代表对传输消息不实施保护；SAID 0x01 代表对数据实施机密性和完整性保护，适用于加密单播控制数据流和单播传输数据流；SAID 0x02 代表对数据流只实施机密性保护，适用于 SS 和 BS 间无保护的传输数据流，剩余的 SAID 标识符由系统所保留。

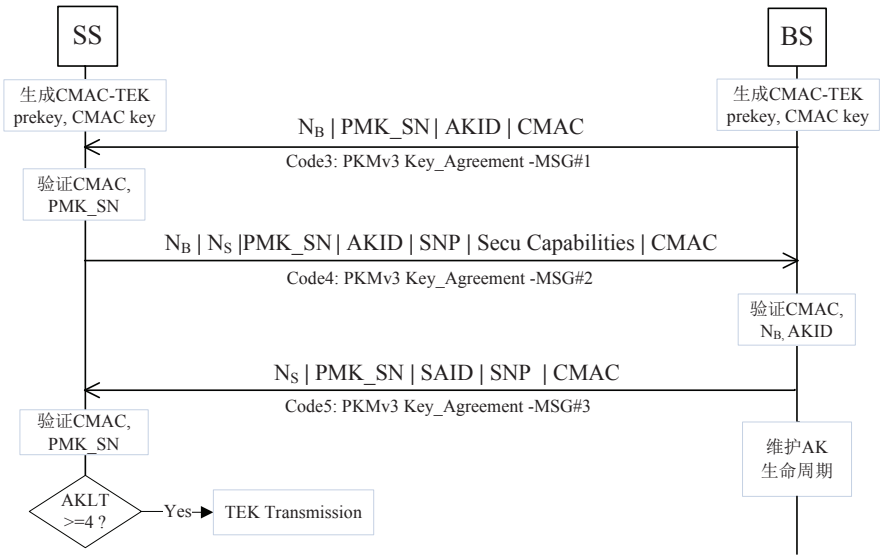

图 3.6: 安全组件协商通信过程

三次握手阶段的三条通信消息如图 3.6 所示，其中的每一条消息同时通过消息验证码实现完整性和认证性保护。BS 首先向 SS 发送安全组件协商信息 Key_Agreement MSG1，包括 BS 在本轮中新生成的随机数 N_B，PMK 的序列号以及 AK 的标识号。AK 标识号 AKID 用于区分不同会话对中生成的 AK，用 Dot16KDF 算法通过 AK 的值，PMK 的序列号，SS 以及 BS 的标识号计算得到。最后在消息末附上由整条明文消息和 CMAC key 生成的 CMAC 消息摘要。

SS 收到 BS 发送的消息后，将明文部分和 CMAC key 计算出的值对比与消息验证码消息，若比较结果一致则发送第二条密钥交换消息作为回应，包括接收到的 BS 发送的随机值 N_B，SS 新生成的随机值 N_S，PMK 的序列号，AK 标识号，安全性协商参数以及 CMAC 消息摘要。安全性协商参数具体包括包序列号的长度以及完整性检验值 ICV 的长度。BS 检验 SS 发送的消息验证码以及随机值 N_B 确认

消息的合法性，并且向 SS 发送随机值 N_S，PMK 的序列号，对应的安全协商参数，SA 标识号 SAID 以及 CMAC 消息验证码。至此，SS 与 BS 间交换并协商了安全通信所需的必要信息。

3.3.3 传输密钥的生成

当完成了三次握手阶段，SS 为每一个合法授权的安全组件标识 SAID 请求加密传输密钥 TEK（Traffic Encryption Key）的关键算子 COUNTER_TEK，并结合 CMAC-TEK perkey 与 SAID 计算出 TEK 的值。COUNTER_TEK 是 BS 在同一个安全组件标识 SAID 下用于生成不同 TEK 而维护的计数器。在 AK 和 AK_COUNT 密钥计数对有效的情况下，每一个协商好的安全组件应该同时拥有两个 TEK，这两个 TEK 由连续的两个 COUNTER_TEK 生成。同时，每当一个新的 CMAC-TEK prekey 生成时，COUNTER_TEK 的值被重置为 0。

BS 向 SS 传输消息的路径称为下行链路，同理，SS 向 BS 传输信息的路径被称为上行链路。已激活的两个 TEK 的选择策略与根据通讯为当前上行还是下行业务有关。若较老的 TEK 失效，则 BS 立刻将其替换为序列号加一的 TEK，并将其用作下行业务的加密密钥。因此，下行业务由较老的 TEK 进行加密，而上行业务则会用较老或较新的 TEK 来解密，在 TEK 整个生命周期的最后阶段，只能通过特定的 TEK 进行加密操作 [56]。

传输密钥请求阶段的消息传输如图 3.7 所示，与三次握手阶段类似，每条消息都通过 CMAC 保护。SS 向 BS 发送传输密钥请求消息 TEK-Request，包括安全组件的标识号，PMK 的序列号以及 CMAC 消息验证码。BS 通过检测消息验证码以及 SAID 的正确性，决定给 SS 发送请求确认消息 TEK-Reply 还是请求无效消息 TEK-Invaid。若请求成功，BS 给 SS 发送安全组件的标识号 SAID，PMK 的序列号，TEK 的计数值 Counter_TEK，加密密钥的序列号以及消息验证码。Counter_TEK 用于与 CMAC-TEK prekey 以及 SAID 一起生成 TEK，而加密密钥序列号则用于区分连续的 TEK，长度为 2bits。若请求失败，BS 则向对应的发送方发送 SAID，相应

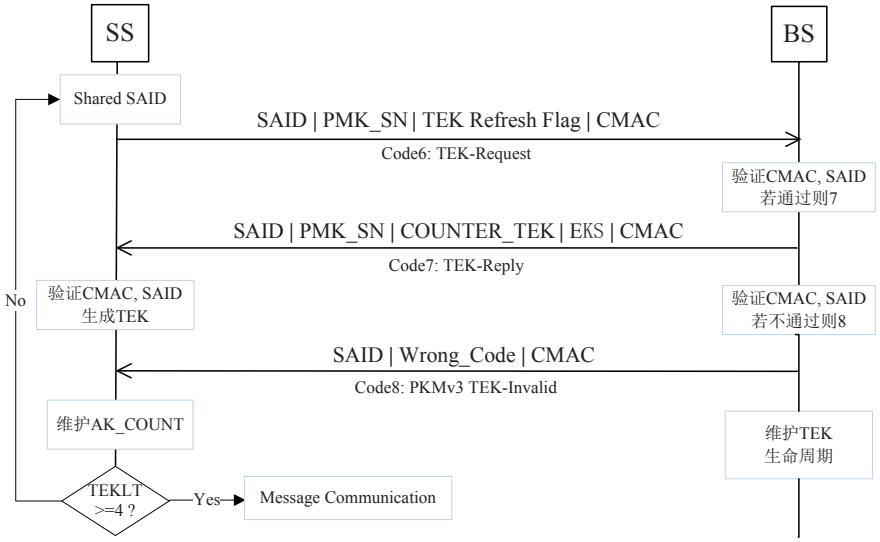

图 3.7: 密钥传输通信过程

的错误代码，错误的原因，以及 CMAC 消息验证码，此时 SS 需向 BS 重新发送 TEK-Request 消息，发起新一轮传输密钥请求。

3.3.4　加密消息传输

经过了完整的 AK 授权，SA 连接，TEK 交换的过程，BS 与 SS 已经建立了安全机制，即可对双方正常通信的内容通过传输密钥 TEK 加密保护，开始消息的安全传输过程。MAC 层中建立协议数据单元 PDU(Protocol Data Unit) 实现消息在该协议层中的传输，发送方和接收方通过映射到该数据单元的 SAID 决定负载的加解密以及认证方式。

实施加密和认证保护的 MAC PDU 的结构如图 3.8 所示，包括通用报头，扩展报头，传输连接负载以及其它安全信息。连接到相同 SA 的负载可以合并封装，通用报头 AGMH 和拓展报头提供了负载的细节信息。安全信息则包括了加密密钥 TEK 的序列号 EKS(Encryption Key Sequence)，包序列号 PN(Package Number) 和完整性校验值 ICV。EKS 和 PN 的长度分别为 2bit 和 22bit，EKS 和 PN 的值通过明文传输，不需要加密。明文负载的内容需要通过已激活的 TEK 进行加密和验证。添加在负载后面的完整性校验值 ICV 的具体长度在 SA 安全组协商过程中由双方决

定是 32bit 还是 64bit。因此，这整个 MAC PDU 的生成过程在原文负载的基础上添加了 7bytes 或者 11bytes 的有效载荷。

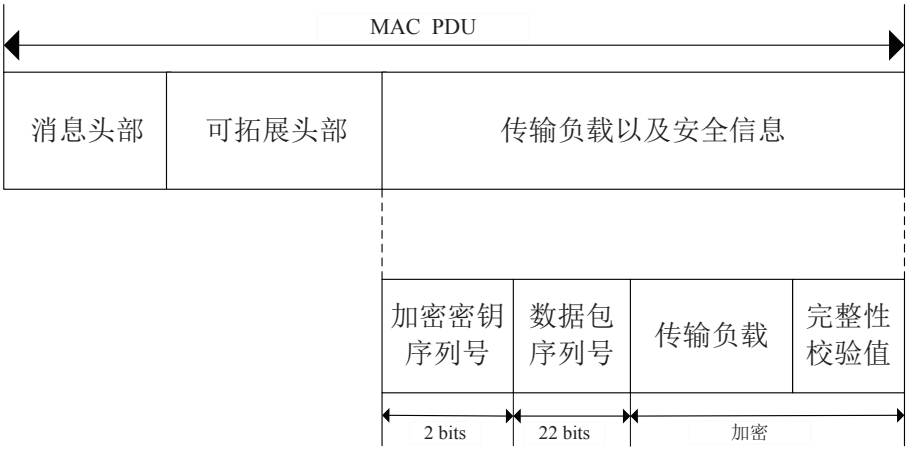

图 3.8: MAC PDU 结构

3.4 PKMv3 密钥重认证

在 PKMv3 协议中，为了保证密钥的活性和安全性，双向认证过程生成的认证密钥 AK 以及密钥传输过程生成的传输密钥 TEK 的生命周期都是有限的，其具体数值由 IEEE802.16m 规范定义 [57]。它们的生命时长会随着消息传输的过程不断减少，直至最终密钥失效。因此，当前 AK 和 TEK 即将到期时，PKMv3 需要启动重新认证的过程，以保持 SS 与 BS 间加密消息的连续传输。

AK 的生命周期是由 BS 维护的系统配置参数。当 AK 生命周期即将到达最后期限时，BS 将收到由 SS 发送的认证请求消息，从而结束当前协议过程并触发新的双向认证过程生成可用的认证密钥 AK。BS 为 SS 提供了连续两个认证密钥 AK，他们之间拥有重叠的生命周期。TEK 的生命周期也是由 BS 维护的系统参数，并且 TEK 的生命周期嵌套在 AK 的生命周期中。在对应的 AK 依然有效的情况下，若 TEK 的生命周期即将到达最终期限，BS 则将收到由 SS 发送的密钥请求消息，从而开始密钥传输过程以生成新的传输密钥 TEK。如图 3.9 所示的密钥分级结构显示了出现在 PKMv3 安全协议中的各密钥和对应密钥通过算法 Dot16KDF 的生成方式，同时也描述了 AK 和 TEK 重认证的执行流程。

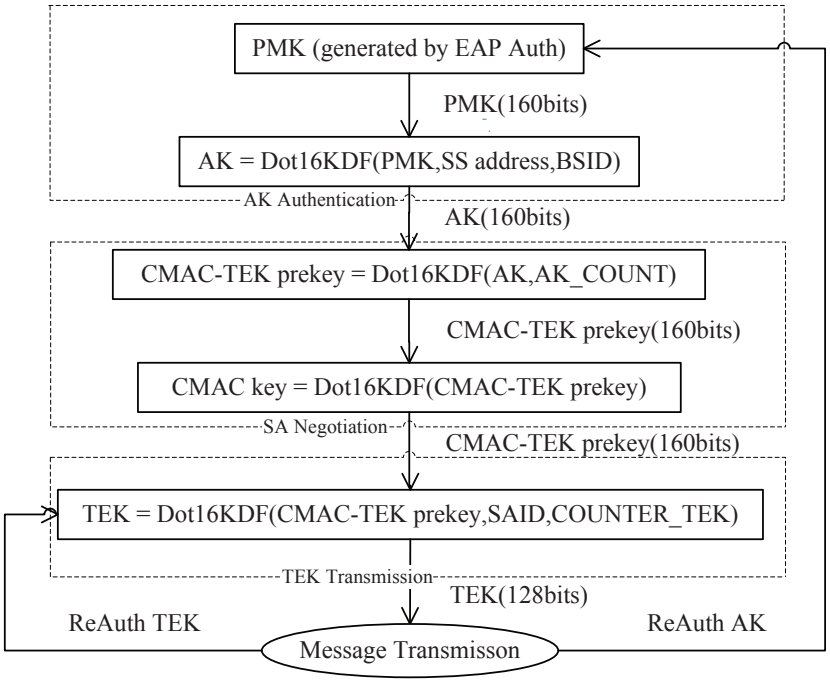

图 3.9: PKMv3 密钥分级结构

3.4.1　AK 重认证

在最初的双向认证阶段，基站 BS 收到非授权终端 SS 发送的第一条认证请求消息，并在认证响应消息中将生成 AK 的关键材料以及其生命时长发送给 SS，触发双方第一个共享 AK 的激活。该 AK 将一直保持有效，并用于生成后续过程的密钥，直到预定义的 AK 生命周期终止。无论何时，基站都能根据终端的请求发送相应 AK 密钥材料。PKMv3 协议中，连续的两代 AK 间具有重复的生命周期，从而保证双方通讯过程的连续进行。

AK 的重认证过程如图 3.10 所示，其中 AK 宽容时间 AGT(AK Grace Time) 是 AK 认证与重认证间隔的最大允许时间，而 AK 激活时间 AAL(AK Active Time) 是预定义的 AK 生命周期。SS 在 AK Grace Time 的开始点进行下一轮的认证请求。而当基站接收到终端新一轮授权请求，从而触发 AK 的切换周期并为其激活下一轮有效 AK。在 PKMv3 协议中，通过 AKsn 来标记连续的 AK，AKsn 是 4 位二进制数值，每生成一个新的认证密钥 AK，AKsn 的数值加一并与 16 取模。第二代 AK 的生命时长为前一代 AK 的剩余生命时长与预定义的 AK 生命定长之和。从而即

使老的 AK 失效，新一代 AK 还能在预定义 AK 生命周期内保持存活。随着老一代 AK 达到 AK Active Time 的结束点，本轮密钥的切换周期便完成了。

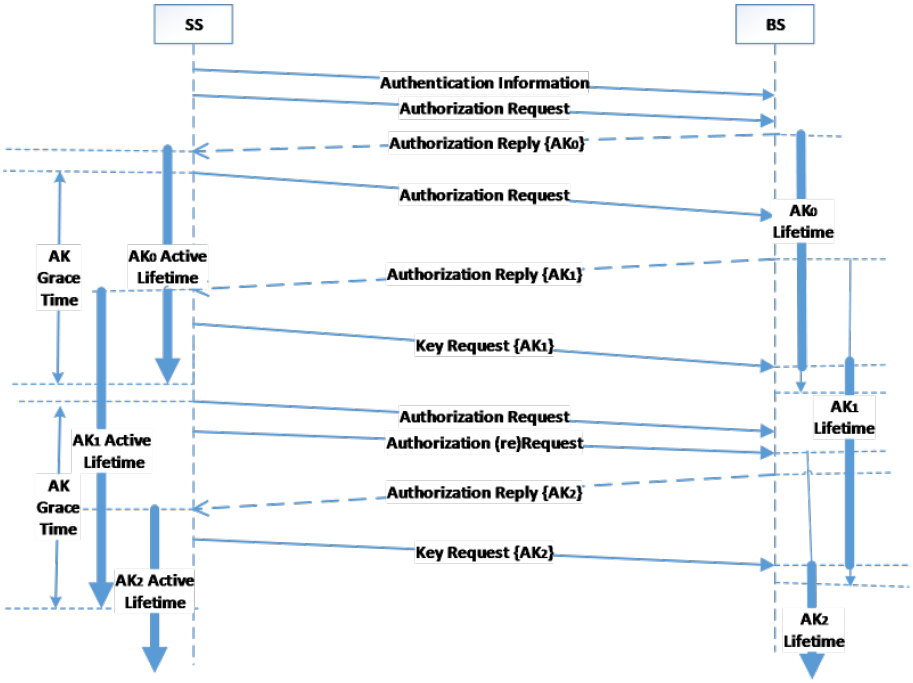

图 3.10: AK 重认证过程

3.4.2　TEK 重认证

基站 BS 为每个 SAID 同时维护两组激活的 TEK，当 SS 与 BS 之间首次执行密钥传输时，双方通过连续的 COUNTER_TEK 生成两个不同的传输密钥。第一个传输密钥的生命周期是预定义 TEK 生命周期的一半，第二个传输密钥以及之后的每个传输密钥的生命周期都等于预定义生命周期的值。在 PKMv3 中，连续的 TEK 通过 TEKsn 标识，TEKsn 由 2 位二进制数值表示，每生成一个新的 TEK 其数值加一并与 4 取模。同一时刻激活的两组 TEK 中 TEK0 用于下行链路的加密，TEK1 用于上行链路的加密。

TEK 的重认证过程是由 TEK 的生命周期耗尽，或者消息传输的数据包序列号 PN 达到最大值触发的。TEK 重认证的具体过程如图 3.11 所示，其中 TEK 宽容时间 TGT(TEK Grace Time) 表示两代 TEK 认证与重认证允许的最大时间间隔，TEK

激活生命时间 TAL(TEK Active Lifetime) 等于 TEK 预定义生命周期。SS 必须在授权宽容时间的期限内获取新的激活 TEK2，并将失效的 TEK0 丢弃。此时，TEK1 被用于下行链路的加密操作，而 TEK2 用于上行链路的加密。在 AK 生命周期有效的条件下，若传输密钥序列号 EKS 的值更新，则 SS 将重复执行 TEK 重认证过程。

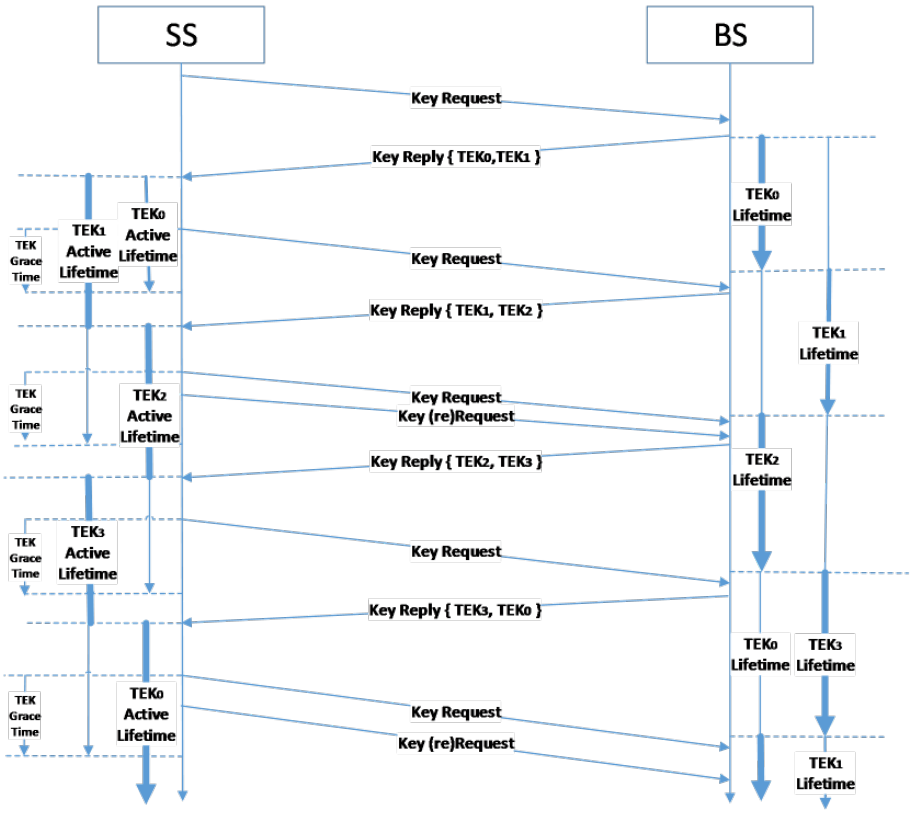

图 3.11: TEK 重认证过程

3.5　本章小结

本章介绍了 IEEE802.16m 标准安全子层中定义的密钥管理 PKMv3 协议。详细描述了无线宽带技术 WiMAX 支持的拓扑结构以及 IEEE802.16m 参考模型中的物理层与介质访问层结构。PKMv3 协议的目的为了是实现手机端与基站间的相互认证以及密钥传输。本章描述了 PKMv3 协议中双向认证过程，安全组件协商过程，密钥生成过程，加密消息传输过程的具体流程。同时，指出该协议中密钥所具有的的时间特性，并描述了各密钥生命周期的迭代过程。

第四章　PKMv3 协议的形式化建模

为了实现对 PKMv3 协议的形式化逻辑推理，需要将该安全协议及其所处的环境视为一个系统，那么在这个系统中，一般包括发送和接受消息的诚实主体和其他攻击者，以及主体间消息的传输。采用 Maude 建模语言将系统中的各主体的性质和行为进行逻辑抽象，从而模拟 PKMv3 协议中双向认证，安全组件协商，密钥生成，加密消息传输，重认证阶段的执行过程，以及入侵者的攻击行为。本文假设 PKMv3 协议中采用的密码算法都是完备的，从而将针对安全协议性质的讨论放在与密码算法的数学细节无关的层次上进行。同时假设基站 BS 在某一时刻，同时只能服务一个手机站 SS，手机站与基站间的通信是同步的，即当发送方发送消息后，必须等待接收方响应后，才能发送下一条消息。

4.1　基本数据类型的定义

4.1.1　通信主体建模

首先，建立函数模块定义 PKMv3 协议中的各抽象数据类型及其相关操作。在函数模块中，参与协议运行的主体被定义为`Station` 数据类型，并定义`Ss` 和`Bs` 类型作为`Station` 类型的子类分别代表手机站和基站。`Ss` 类型的数据通过输入参数为`String` 预定义类型的构造函数`ss` 生成，如`ss(“a”)` 代表标识符为 a 的手机站，同理，`Bs` 类型的数据通过输入参数为`String` 预定义类型的构造函数`bs` 生成，如`bs(“b”)` 代表标识符为 b 的基站。

```
1 sorts Station Ss Bs . subsorts Ss Bs < Station .
2 op ss : String -> Ss [ctor] .
3 op bs : String -> Bs [ctor] .
```

主体拥有生成随机数的能力，将 SS 和 BS 各自的初始值用构造函数seed和seed1 表示，数据类型定义为Rand。next 操作用于生成随机数序列，如 SS 的随机数序列可表示为seed，next(seed)，next(next(seed))...。为了区分网络中各主体发送的随机值，将随机值Nonce 通过nonce 构造函数建模，三个输入参数分别代表发送消息的主体，接收消息的主体以及本次消息所使用的随机数。操作符equals 表示两个随机值相等，当且仅当它们的发送和接收主体，以及随机数的数值相等，若比较结果相等，则返回布尔值 true，否则为 false，其中两个随机数的比较通过equal 操作实现。

```
sorts Rand Nonce .
op seed : -> Rand [ctor] . op seed1 : -> Rand [ctor] .
op next : Rand -> Rand . op equal : Rand Rand -> Bool [comm] .
op nonce : Station Station Rand -> Nonce [ctor] .
vars S1 S2 S3 S4 : Station . vars R1 R2 : Rand .
eq equal(R1,R1) = true . eq equal(seed,next(R1)) = false .
eq equal(next(R1),next(R2)) = equal(R1,R2) .
ceq equals(nonce(S1,S2,R1),nonce(S3,S4,R2)) = true
    if S1 == S3 /\ S2 == S4 /\ R1 == R2 .
```

协议中的每个主体在出厂时便带有属于自己的一对公钥和私钥，公钥存放在公钥基础设施 PKI(Public Key Infrastructure) 中，网络中的任意主体均可获取。私钥由主体自身持有，其它任意主体无法获取。定义密钥类型Key 代表协议中出现的所有的密钥，并定义公钥类型Pubkey 和私钥类型Prikey 为它的子类。操作符pubkey 以及prikey 以主体名作为输入参数，生成各主体的公钥与私钥。判定某个私钥与公钥是否互相匹配通过pair 操作实现，利用其中的getstation 函数提取出密钥中的主体，从而判断出公私密钥中的主体是否为同一个，若主体相同，pair 操作则返回布尔值 true。

```
sorts Key Pubkey Prikey . Subsorts Pubkey Prikey < Key .
op pubkey : Station -> Pubkey [ctor] .
op prikey : Station -> Prikey [ctor] .
op getstation : Pubkey -> Station .
op getstation : Prikey -> Station [ditto] .
op pair : Pubkey Prikey -> Bool .
vars S3 S4 : Station .
var PUB1 : Pubkey . var PRI1 : Prikey .
eq getstation(pubkey(S3)) = S3 . eq getstation(prikey(S4)) = S4 .
ceq pair(PUB1,PRI1) = true if getstation(PUB1) == getstation(PRI1) .
```

此外，各主体的 MAC 地址声明为数据类型MacAddr，并通过输入参数为 Station 类型的macaddr 操作构造。各主体用于识别身份的 X.509 证书由 CA(Certificate Authority) 认证中心签发，抽象为Cert 数据类型，由证书的主体、该主体的 MAC 地址以及公钥三个参数通过cert 操作构造生成。当网络中的主体接收到其他主体的证书时，必须能从其证书中提取出相应的信息，因此操作符getstat，getaddr，getpub 分别用于从证书中提取出主体，MAC 地址以及公钥信息。

```
sorts Cert MacAddr .
op macaddr : Station -> MacAddr .
op cert : Station MacAddr Pubkey -> Cert .
op getstat : Cert -> Station .
op getaddr : Cert -> MacAddr .
op getpub : Cert -> Pubkey .
var S5 : Station .
eq getstat(cert(S5,macaddr(S5),pubkey(S5))) = S5 .
eq getaddr(cert(S5,macaddr(S5),pubkey(S5))) = macaddr(S5) .
eq getpub(cert(S5,macaddr(S5),pubkey(S5))) = pubkey(S5) .
```

4.1.2　消息内容建模

为了便于管理，将网络中出现的主体类型Station，密钥类型Key，证书类型Cert，随机值类型Rand 等划分为内容类型Content 的子类。Content 类型的数据满足交换律和结合律，它的单位元是contnil。Content 类型数据的集合为内容集合Contents 类型，并定义其为Content 的父类，集合中各内容间用符号“,”隔开。操作符inc 用于判断符号左边的内容是否属于右边的内容集合，若内容集合中包含该内容，则返回布尔值 true，否则返回 false。

```
var Content Contents .
subsorts Station Key Cert Nonce Rand < Content .
subsort Content < Contents .
op contnil : -> Contents [ctor] .
op _,_ : Contents Contents -> Contents [assoc comm id: contnil] .
op _inc_ : Content Contents -> Bool .
var C : Content . var CS : Contents .
eq CS,CS = CS .
eq C inc (C,CS) = true .
eq C inc CS = false [owise] .
```

在双向认证阶段中，存在对信息的加密，数字签名，以及校验和处理。加密操作用于保护消息的机密性，利用主体的公钥通过encrypt 操作将Contents 类

型的信息集加密为Cipher 类型的密文。带条件decrypt 操作定义了仅当持有相同主体的私钥，才能够将加密内容解密。对信息的数字签名操作保障信息的完整性以及来源的可靠性，sencrypt 操作用主体自身的私钥对消息加密实现数字签名，sdecrypt 操作定义只有通过相同主体的公钥，才能将数字签名解密。校验和（Checksum）用于检测消息的完整性是否在传输过程中遭到破坏。通过校验和算法checksum 将Contents 类型的信息和初始向量iv 作为输入参数，从而计算出Checksum 类型的完整性校验和，并一般附在明文消息的最后。

```
subsorts Cipher Ncipher < Content .
op encrypt : Contents Pubkey -> Cipher .
op decrypt : Cipher Prikey -> Ncipher .
op sencrypt : Contents Prikey -> Content .
op sdecrypt : Contents Pubkey -> Content .
op checksum : Contents Iv -> Checksum [ctor] .
op iv : -> Iv [ctor] .
var Pub1 : Pubkey . var Pri1 : Prikey . var C1 : Contents .
ceq decrypt(encrypt(C1,PUB1),PRI1) = C1
    if getstation(PUB1) = getstation(PRI1) .
ceq sdecrypt(sencrypt(C,Pri1),Pub1) = C
    if getstation(Pri1) = getstation(Pub1) .
```

4.1.3 密钥信息建模

在 PKMv3 协议中使用的 Dot6KDF 密钥生成算法，通过构造函数algo 抽象表示，从而使得实验建模与密钥算法的具体实现无关。双向认证过程中生成的主密钥 PMK 用于生成认证密钥 AK，通过pmk 操作构造生成，各输入参数分别代表参与会话的手机站 SS、基站 BS、各自在认证阶段发送的随机值以及密钥算法。同时，主密钥序列号 PMK_SN 通过pmksn 函数以 PMK 为输入参数生成。

```
sorts Pmk Pmksn Algo. subsort Pmk < Key .
op algo : -> Algo [ctor] .
op pmk : Ss Bs Nonce Nonce Algo -> Pmk [ctor ] .
op pmksn : Pmk -> Pmksn [ctor] .
```

SS 的加密能力使用capa 操作声明，并定义为Capa 数据类型。同时，Ssaddr 代表手机端的物理地址，该类型的数据通过操作符ssaddr 生成。认证密钥 AK 由 PMK 衍生而成，通过ak 操作计算生成，输入参数包括 PMK、参与的基站、手机端的物理地址以及密钥生成算法。数据类型Akcount 代表基站为 AK 维护的计数器

AK_COUNT，为了区分不同网络连接中的 AK_COUNT 初值，以传输随机值为输入参数通过akcount 操作计算生成。AkId 则代表 AK 的标识符，通过 AK、PMK 序列号、参与会话的手机端与基站、密钥生成算法为算子计算得出。最后，将Ak、AkId、Akcount 等数据类型归类为PmkInfo 类型，并定义Ssaddr、Capa、PmkInfo 类型均属于消息内容Content 类型。

```
sorts Ak Akcount AkId Ssaddr PmkInfo Capa .
subsorts Ak AkId Akcount Pmk Pmksn < PmkInfo .
subsorts Ssaddr Capa PmkInfo < Content .
op capa : Station -> Capa .
op ssaddr : Station -> Ssaddr .
op ak : Pmk Bs Ssaddr Algo -> Ak [ctor] .
op akcount : Nonce Nonce -> Akcount [ctor] .
op akid : Ak Pmksn Ss Bs Algo -> AkId [ctor] .
```

CMAC-TEK prekey 用于生成 CMAC key 以及之后的传输密钥 TEK，并定义为 CmacTek 类型，使用 cmactek 函数以 AK，AK 计数值和密钥算法为输入参数生成。CMAC key 是用于计算消息验证码的密钥，定义为Cmackey 数据类型，通过cmackey 操作以CMAC-TEK prekey 和密钥算法为输入参数生成。消息验证码由消息原文和 CMAC 密钥计算得出，由cmac 操作计算生成Cmac 数据类型的结果。同时，将 CMAC 相关数据类型定义为Content 类型的子集。在实际协议过程中，需要对比接收到的消息验证码与计算得出的 CMAC 值是否一致，因此定义equalm 操作比较两个消息摘要 CMAC 的值是否一致。比较结果返回 true 当且仅当它们的消息原文和 CMAC 密钥均为一致，其中getcmac 函数以及getkey 函数用于从消息验证码中提取消息原文和密钥。

```
sorts CmacTek Cmackey Cmac.
subsorts CmacTek Cmackey Cmac < Content .
op cmactek : Ak Akcount Algo -> CmacTek .
op cmackey : CmacTek Algo -> Cmackey .
op cmac : Contents Cmackey -> Cmac .
op equalm : Cmac Cmac -> Bool [comm] .
op getcmac : Cmac -> Contents .
op getckey : Cmac -> Cmackey .
var C1 : Contents . var CKEY1 : Cmackey . vars CM1 CM2 : Cmac .
eq getcmac( cmac(C1,CKEY1) ) = C1 .
eq getckey( cmac(C1,CKEY1) ) = CKEY1 .
ceq equalm(CM1,CM2) = true if getcmac(CM1) == getcmac(CM2) /\ getckey(CM1) ==
    getckey(CM2) .
```

主体间协商的安全组件信息包括各自支持的加密算法能力和安全协商参数。各主体的加密算法能力通过secucapa 操作以站点为输入参数生成SecuCapa 类型的数据。安全协商参数定义为SNP 数据类型，由代表完整性校验值长度的ICVsize 数据类型和代表数据包大小的PN 数据类型组成，其中完整性校验值的长度以及数据包的大小都通过自然数类型Nat 表示。SS 和 BS 间协商好的安全组件通过 SAID 表示，并定义为SaId 数据类型，通过said 操作以安全组件协商阶段发送的随机值和加密算法能力为参数建模。同时，将安全组件相关数据类型SecuCapa，SNP 和SaId 定义为Content 类型的子集。

```
sorts SaInfo SaId SecuCapa SNP ICVsize PN.
subsorts SecuCapa SNP SaId < SaInfo .
subsort SaInfo < Content .
subsorts Nat < ICVsize PN .
op secucapa : Station -> SecuCapa .
op snp : ICVsize PN -> SNP .
op said : Nonce Nonce SecuCapa -> SaId .
```

传输密钥 TEK 用于加密双方后续的消息传递，TEK 序列号定义为 EKS 数据类型，其初始值使用构造函数eks 生成，之后的 TEK 序列号由nexte 函数生成。COUNTER_TEK 是 BS 为 TEK 维护的计数器，用于在相同安全组件下生成不用的 TEK，它的初值由安全组件协商阶段发送的随机值生成，并定义为CounterTek 数据类型，之后的 TEK 计数器序列通过nextc 函数生成。TEK 刷新标识定义为TRFlag 类型，flag(0) 表示首次生成 TEK，而flag(1) 表示更新 TEK。操作符pnexpire 和icverror 生成 TEK 请求的错误代码，分别表示由数据包达到数量上限和完整性校验值错误导致的请求失败。在相同安全组件下的同一时刻，BS 为 SS 维护了两组 TEK，tek1 和tek2 操作生成这两组 TEK，通过 CMAC-TEK prekey，安全组件标识符，COUNTER_TEK 和密钥算法作为输入参数。同时，定义 TEK 相关数据类型和错误代码类型为Content 类型的子集。

```
sorts TekInfo WrongCode CounterTek EKS TEK TRFlag .
subsorts TEK CounterTek EKS TRFlag < TekInfo .
subsorts WrongCode TekInfo < Content .
op eks : -> EKS [ctor] . op nexte : EKS -> EKS .
op countertek : Nonce Nonce -> CounterTek .
op nextc : CounterTek -> CounterTek .
op flag : Nat -> TRFlag .
```

```
op pnexpire : -> WrongCode . op icverror : -> WrongCode .
ops tek1 tek2 : CmacTek SaId CounterTek Algo -> TEK .
```

4.1.4　传输消息建模

最后，通过构造函数实现 PKMv3 协议中各阶段双方传输消息的建模。在双向认证阶段中，SS 与 BS 间通过 EAP 认证传递三条消息，包括定义为`AuthRequest` 类型的认证请求消息，定义为`AuthResponse` 类型的认证响应消息，定义为`Auth-Confirm` 类型的认证确认消息。在安全组件协商阶段，BS 与 SS 间通过三次握手交换了三条消息，包括定义为`ThreeWayRequest` 类型的三次握手请求消息，定义为`ThreeWayResponse` 类型的三次握手响应消息，定义为`ThreeWayConfirm` 类型的三次握手确认消息。在传输密钥的生成阶段，SS 和 BS 间传输了密钥请求和密钥响应消息，密钥请求消息定义为`TekRequest` 类型，密钥响应消息分为两类，若请求成功则响应`TekResponse` 类型的消息，若请求失败则响应`TEKInvalid` 类型的消息。加密消息传输阶段各消息内容都通过 TEK 加密保护，由于每条消息的内容组成都是不确定的，因此不为其定义特定的构造函数。描述 PKMv3 协议中各消息构造函数的输入参数所表示的内容如图 4.1 所示。

AK Authentication:
Msg1. SS -> BS: [$Cert_{SS}$ | N_S | Capabilities]SIG_{SS}(1)
Msg2. BS -> SS: [$Cert_{BS}$ | N_S | N_B | ENC_{SS}(PMK) | PMK_SN]SIG_{BS}(2)
Msg3. SS -> BS: N_B | SS_Address | CheckSum(3)
SA Negotiation:
Msg4. BS -> SS: N_B | PMK_SN | AKID | CMAC(4)
Msg5. SS -> BS: N_B | N_S |PMK_SN | AKID | SNP | Security Capabilities | CMAC(5)
Msg6. BS -> SS: N_S | PMK_SN | SAID | SNP | CMAC(6)
TEK Transmission:
Msg7. SS -> BS: SAID | PMK_SN | TEK Refresh Flag | CMAC(7)
Msg8a. BS -> SS: SAID | PMK_SN | COUNTER_TEK | EKS | CMAC(8a)
Msg8b. BS -> SS: SAID | Wrong_Code | CMAC(8b)
Message Communication:
Msg9. BS <-> SS: EKS | PN | Communication Contents | ICV | TEK(9)

图 4.1: PKMv3 消息传输内容

```
sorts AuthRequest AuthResponse AuthConfirm .
subsorts AuthRequest AuthResponse AuthConfirm < Content .
op authrequest : Cert Nonce Capa -> AuthRequest [ctor] .
op authresponse : Cert Nonce Nonce Cipher Pmksn -> AuthResponse [ctor] .
op authconfirm : Nonce Ssaddr -> AuthConfirm [ctor] .
sorts ThreeWayRequest ThreeWayResponse ThreeWayConfirm .
subsorts ThreeWayRequest ThreeWayResponse ThreeWayConfirm < Content .
op threewayrequest : Nonce Pmksn AkId -> ThreeWayRequest [ctor] .
op threewayresponse : Nonce Nonce Pmksn AkId SNP SecuCapa -> ThreeWayResponse [
    ctor] .
op threewayconfirm : Nonce Pmksn SNP SaId -> ThreeWayConfirm [ctor] .
sorts TekRequest TekResponse TekInvalid .
subsorts TekRequest TekResponse TekInvalid < Content .
op tekrequest : SaId Pmksn TRFlag -> TekRequest [ctor] .
op tekresponse : SaId Pmksn CounterTek EKS -> TekResponse [ctor] .
op tekinvalid : SaId WrongCode -> TekInvalid [ctor] .
```

4.2 网络环境的建模

4.2.1 系统状态的形式化定义

PKMv3 网络环境的系统状态组成如图 4.2 所示。各站点及其收集的信息集合构成网络中的结点，各网络结点向消息池中发送消息，目标结点再从消息池中获取消息。传输消息的 SS 和 BS 间建立连接，并共同维护连接状态参数，参数包括双方共享的 AK 生命周期，TEK 生命周期，以及当前所处的通信阶段。在网络中，还存在其它诚实主体构成的结点，以及入侵者结点，入侵者从消息池中窃取消息，或向目标结点投放重放消息或伪造的消息。

网络中的主体结点定义为`Node` 类型，并用`node[_]:_` 构造函数表示，其中下划线表示各参数的位置，第一个参数为该节点的主体名字，第二个参数为该主体已生成或者已得到的信息集。网络中传播的消息定义为`Message` 类型，网络中若无消息则用操作符`msgnil` 表示。主体发送消息通过构造函数`from_to_send_` 实现，代表第一个输入参数表示的主体发送消息给第二个参数表示的主体，消息的具体内容为第三个参数表示的内容集合。定义数据类型为`Process` 的构造函数`pak`，`psa`，`ptek` 和`pmt` 代表当前执行的通信阶段分别为双向认证过程、安全组件协商过程、传输密钥生成过程以及加密消息传输过程。AK 和 TEK 的生命周期

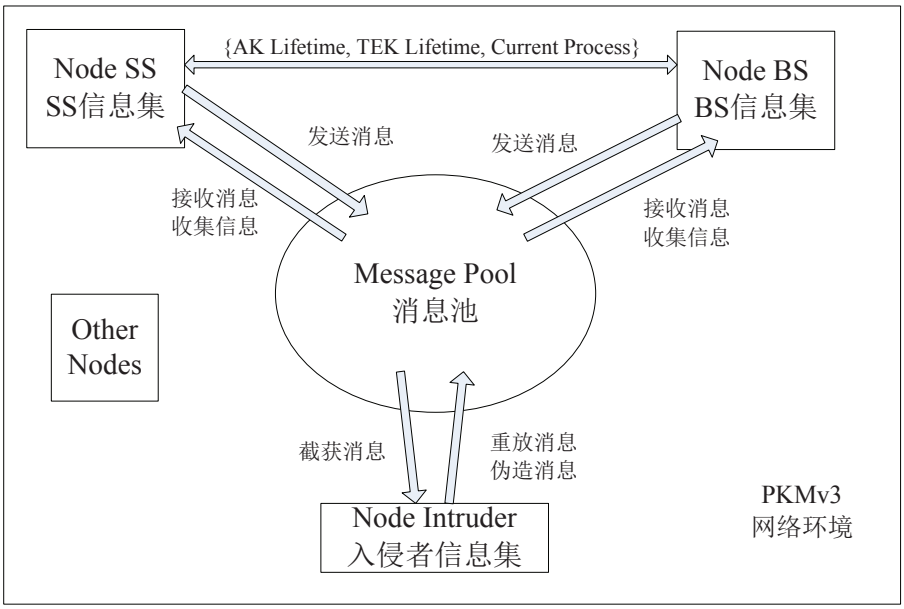

图 4.2: PKMv3 系统状态组成

由自然数表示，定义为生命周期数据类型`LTime`，并使用操作符`agt`，`aat` 申明 AK 的宽容时间和生命时长，以及操作符`tgt`，`tat` 申明 TEK 的宽容时间和生命时长。由此，进行信息交互的一组 SS 和 BS 间的连接状态可通过三元组`{_,_,_}`表示，其中各参数分别为 AK 的剩余生命周期、TEK 剩余的生命周期、以及当前通信所处的阶段，并被定义为`System` 类型。

```
sort Message Node LTime Process Systime .
subsort Nat < LTime .
op node[_]:_ : Station Contents -> Node [ctor] .
op msgnil : -> Message [ctor] .
op from_to_send_ : Station Station Contents -> Message [ctor] .
ops agt aat tgt tat : -> LTime .
ops pak psa ptek pmt : -> Process .
op {_,_,_} : LTime LTime Process -> Systime .
```

PKMv3 网络的系统状态定义为`States` 类型，系统状态中包括各主体节点，消息传输，以及节点间的连接参数，从而将`Node` 类型、`Message` 类型、以及`System` 类型定义为`States` 类型的子集，并定义`__` 操作实现各部分子状态的连接，连接操作满足结合律和交换律。为了在系统验证过程中检测某个子状态是否属于系统状态，定义`ina` 操作返回符号左边的子状态是否为右边的系统状态的子集，若满足条件则返回布尔值 true，否则返回 false。

```
sort States .
subsorts Message Node Systime < States .
vars S1 S2 : States .
op __ : States States -> States [assoc comm] .
op ina : States States -> Bool .
eq S1 S1 = S1 .
eq S1 ina (S1 S2) = true .
eq S1 ina S2 = false [owise] .
```

4.2.2 消息传输形式化建模

为了描述协议中主体通信的动态过程，采用重写规则模拟认证过程中各消息的发送和接收，从而实现 PKMv3 网络系统的状态转换。本论文中总共定义的重写规则条数及其分类如表 4.1 所示。由于篇幅限制，本文将不逐条描述，通过各协议阶段中具有代表性的规则为例，阐述 PKMv3 通讯过程的实验原理。重写规则中各变量的数据类型以及本章缺省的重写规则可参考附录 A。

表 4.1: 各阶段重写规则条数

站点	AK 认证	SA 交换	TEK 生成	消息传输	重认证
SS/BS	6	6	9	1	2
Intruder	9	3	3	2	0

在建模过程中，变量 A 代表Ss 数据类型的手机端，变量 B 代表Bs 数据类型的基站。为了简化建模代码，通过成员关系关键字mb 定义主体 A 的数字证书为CertA 类型，同理定义主体B 的数字证书为CertB 类型，由此规则中非首次出现的 A 和 B 数字证书便可由变量CERT_A 和CERT_B 匹配。

```
sorts CertA CertB . subsorts CertA CertB < Cert .
var CERT_A : CertA . var CERT_B : CertB .
mb cert( A,macaddr(A),pubkey(A) ) : CertA .
mb cert( B,macaddr(B),pubkey(B) ) : CertB .
```

双向认证通信过程

在 SS 申请认证密钥 AK 之前，需要先向 BS 发送自己的数字证书以实现整个系统的初始化。BS 无需对初始化消息做出回应。在这基础上，SS 向 BS 发送认证

请求消息作为双向认证阶段的开始。顺利实施的双向认证过程最终生成认证密钥 AK，并利用其生成后续阶段的密钥。之前的章节已提到过 AK 具有有限的生命时长，在当前 AK 失效前 SS 需向 BS 请求重认证过程。在整个 PKMv3 阶段 AK 的生命周期由图 4.3 所示的 AK 状态机描述。用于描述 AK 序列号的 AKsn 取值范围为 0 到 15，因此 AK 的序列号依次加一并与 16 取模。

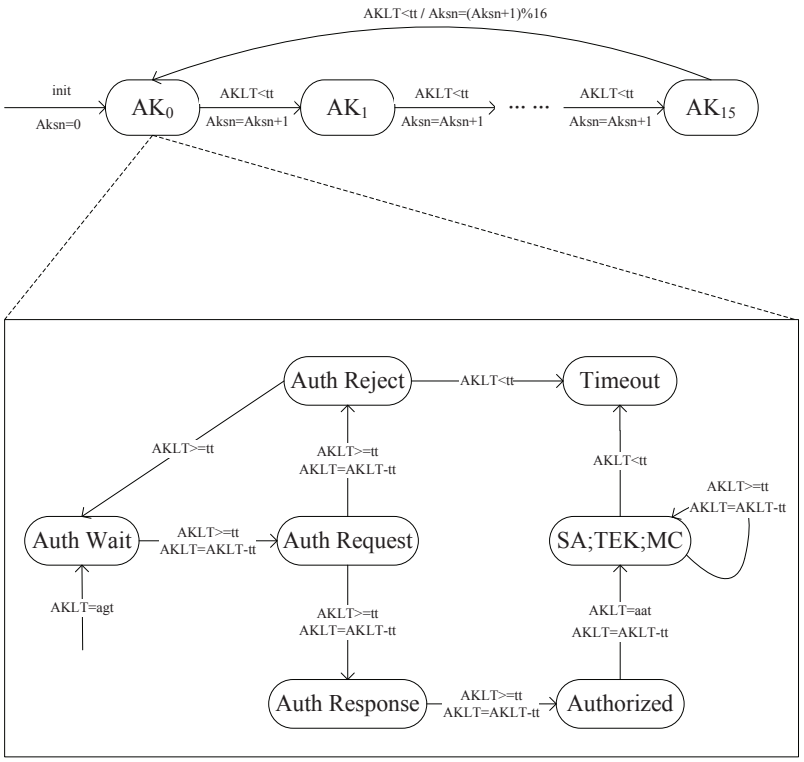

图 4.3: AK 状态机

在 AK 认证或重认证的初始时刻，AK 状态机处于 Auth Wait 状态，同时 AK 的生命周期AKLT 变量被赋值为 AK 宽容时间agt，即规定 SS 必须在agt 定长时间内获取并生成 AK，否则需要重新开始认证过程。此时若AKLT 的数值大于消息传输时长tt，则发送认证请求消息到达 Auth Request 状态，每发送一次消息，AKLT 的值都减少tt 时长。若 SS 认证请求成功且AKLT 的值大于tt，BS 向 SS 发送认证确认消息到达 Auth Response 状态，并且 AK 宽容时间的剩余时长同样减少tt。此时认证时长若仍在agt 范围内，SS 向 BS 发送认证确认消息，状态机到达 Authorized 状态，此时双方均生成共享的认证密钥 AK。一旦新的 AK 被激活，

其生命周期AKLT 被赋值为 AK 生命时长定值aat，并可用于后续 SA 协商阶段，TEK 生成阶段，消息传输阶段的消息传输和密钥生成。在这些阶段若 AK 剩余生命时长大于消息传输时长tt，每发送依次消息AKLT 的值减小tt，直至最终当前 AK 失效达到 TimeOut 状态，从而开始下一轮新的 AK 的请求过程。在传输密钥生成阶段之前，TEKLT 的值一直被设为初值 0。

标签为SendAuthRequestMsgReal 的重写规则定义了 SS 向 BS 发送认证请求消息从而启动认证或重认证的过程。在本规则中，消息发送前 SS 和 BS 的消息集中拥有自身的随机数R0 和R1，以及 X.509 证书，并且此时 AK 生命周期AKLT 被赋值为 AK 宽容时间agt。规则匹配后 SS 生成随机值nonce(A,B,R0) 并添加到 SS 本地信息集中。发送的认证请求消息由authrequest 构造函数生成，包括代表 SS 数字证书的变量CERT_A，SS 本轮生成的随机值nonce(A,B,R0) 以及自身加密能力capa(A)，并通过sencrypt 操作实现消息的数字签名。AK 宽容时间通过sd 自然数减法操作减少消息传输时间tt，TEK 此时的生命周期处于初始状态，当前执行阶段保持为代表双向认证阶段的pak 标识。

```
rl [SendAuthRequestMsgReal] :
  (node[A]: R0,CERT_A) (node[B]: R1,CERT_B)
  (msgnil) {agt,TEKLT,pak}
 =>
  (node[A]: next(R0),CERT_A,nonce(A,B,R0)) (node[B]: R1,CERT_B)
  (from A to B send sencrypt(authrequest(CERT_A,nonce(A,B,R0),capa(A)),prikey(A
      ))) {sd(agt,tt),TEKLT,pak} .
```

重写规则RecvAuthRequestMsgReal 描述了 SS 从信息池中接收 BS 发送的认证响应消息的过程。在重写规则中，规则左边状态中MSG2 变量代表 BS 发送给 SS 的认证响应消息，并且该消息由sencrypt 操作实现数字签名。此时 SS 信息集中包括其自身的随机数，数字证书和生成的随机值。If 条件语句阐明若通过getnon22 操作从MSG2 提取出的随机值与 SS 生成的随机值相等，同时若sdecrypt 操作使用的 BS 公钥能够解密消息的数字签名，SS 则从MSG2 中获取相关信息添加到信息集中，其中getcert2 操作提取出 BS 数字证书，getnon22 操作提取出 BS 生成的随机值，getcipher 操作提取出 PMK 加密内容，并通

过decrypt 操作进行解密，getpmksn2 操作提取出 PMK 序列号。认证确认消息成功被 SS 接收后，消息池转换为空消息 msgnil 状态。

```
crl [RecvAuthResponseMsgReal] :
  (node[A]: next(R0),CERT_A,nonce(A,B,R0))
  (from B to A send sencrypt(MSG2,PRI2)) {AKLT,TEKLT,pak}
 =>
  (node[A]: next(R0),CERT_A,nonce(A,B,R0),getcert2(MSG2),getnon22(MSG2),
       decrypt(getcipher(MSG2),prikey(A)),getpmksn2(MSG2))
  (msgnil) {AKLT,TEKLT,pak}
  if equals(getnon21(MSG2),nonce(A,B,R0)) == true /\
    getpub(getcert2( sdecrypt(sencrypt(MSG2,PRI2),pubkey(B)))) == pubkey(B) .
```

安全组件协商通信过程

SS 和 BS 互相成功认证后，双方均通过ak 构造函数生成共享认证密钥 AK，并且 BS 为激活的 AK 维护计数器 AK_COUNT。SendThreeWayRequestReal 描述了 BS 向 SS 发送三次握手请求消息以协商安全组件的过程。本条规则中 if 条件语句表示规则匹配条件为当前 AK 为有效状态，并且 SS 仍未生成 AKID。若满足条件，BS 删除消息集中认证阶段随机值，并生成本轮新的随机值。与此同时，通过操作akid 生成 AK 标识符 AKID，通过cmactek 操作生成 CMAC-TEK prekey，以及通过cmackey 操作生成 CMAC key。函数threewayrequest 构造三次握手请求消息，输入参数包括 BS 随机值nonce(B,A,R1)，变量PMKSN 代表的 PMK 序列号，生成的 AKID。消息验证码由cmac 操作生成，并与请求消息一同发送给 SS。双方连接状态中的当前执行阶段由pak 标识转换为psa 标识。

```
crl [SendThreeWayRequestMsgReal] :
  (node[B]: next(R1),nonce(B,A,R1),PMKSN,AKCOUNT,AK,CS1)
  (msgnil) {AKLT,TEKLT,pak}
 =>
  (node[B]: next(next(R1)),nonce(B,A,next(R1)),PMKSN,AKCOUNT,AK,
       akid(AK,PMKSN,A,B,algo),cmactek(AK,AKCOUNT,algo),
       cmackey(cmactek(AK,AKCOUNT,algo),algo))
  (from B to A send (threewayrequest(nonce(B,A,next(R1)),PMKSN,
      akid(AK,PMKSN,A,B,algo)),cmac(threewayrequest(nonce(B,A,next(R1)),PMKSN,
      akid(AK,PMKSN,A,B,algo)),cmackey(cmactek(AK,AKCOUNT,algo),algo))) )
  {sd(AKLT,tt),TEKLT,psa}
  if AKLT >= tt /\ not(akid(AK,PMKSN,A,B,algo) inc CS1) .
```

重写规则RecvThreeWayResponseReal 描述了 BS 接收 SS 发送的三次握手响应消息的过程。在本条规则的状态转换前，消息池中存在 SS 发给 BS 的三

次握手响应消息 MSG5 以及消息验证码。规则中的 if 条件语句声明若从MSG5 中获取的 AKID 和随机值与 BS 信息集中的 AKID 和随机值相等，并且满足cmac 操作计算出的消息摘要与接收到的消息验证码相等，则 BS 从消息池中提取出该消息。从而 BS 利用getnon52，getsnp5 和getsc 操作从 MSG5 中分别提取出 SS 新生成的随机值，安全协商参数以及 SS 支持的安全算法。当前执行阶段标识符保持psa 不变，接收消息不消耗 AK 的生命时长。

```
crl [RecvThreeWayResponseReal] :
  (node[B]: nonce(B,A,R1),AKID,CMACKEY,CS1)
  (from A to B send (MSG5,CMAC2)) {AKLT,TEKLT,psa}
 =>
  (node[B]: nonce(B,A,R1),AKID,CMACKEY,CS1,
       getnon52(MSG5),getsnp5(MSG5),getsc(MSG5))
  (msgnil) {AKLT,TEKLT,psa}
  if getakid5(MSG5) == AKID /\ equals(getnon51(MSG5),nonce(B,A,R1))
    /\ equalm(cmac(MSG5,CMACKEY),CMAC2).
```

传输密钥生成通讯过程

TEK 的生命周期嵌入在 AK 的生命周期中，TEK 的认证和重认证过程当且仅当相应的 AK 处于有效状态。在整个 PKMv3 阶段 TEK 的生命周期由图 4.4 所示的 TEK 状态机描述。标识连续 TEK 的序列号 TEKsn 取值范围为 0 到 3，在相同的安全组件下，每生成一个新的 TEK 数值加一并与 4 取模。TEK 的生命周期通过TEKLT 变量表示，在安全组件交换的结束状态，TEKLT 被赋值为 TEK 宽容时间tgt，即限定传输密钥生成过程的最大使用时间，否则需要重新触发密钥生成阶段。SS 通过向 BS 发送密钥请求消息作为传输密钥生成过程的开始，此时 TEK 状态机由 TEK Wait 状态到达 TEK Request 状态，并将AKLT 和TEKLT 的值减少消息传输时间tt。若密钥请求成功且对应 AK 和 TEK 均为有效状态，BS 则向 SS 发送密钥响应消息达到 TEK Response 状态，同理将AKLT 和TEKLT 的值减少消息传输时间tt。若密钥请求失败且对应 AK 和 TEK 均为有效状态，SS 则向 BS 发送请求失败消息到达 TEK Reject 状态，此时 SS 需要重新开始新一轮的密钥请求，从而回到 TEK Wait 状态。成功生成 TEK 的 SS 和 BS 可以通过该 TEK 加密后续消息内容，从而TEKLT 被赋值为 TEK 生命定长tat。双方每传输一条加密消息，相应的

TEK 和 AK 的剩余消息减少tt。若消息传输过程中 TEK 先失效，则开始下一轮的 TEK 请求过程。若 AK 先失效，重启整个 PKMv3 过程。

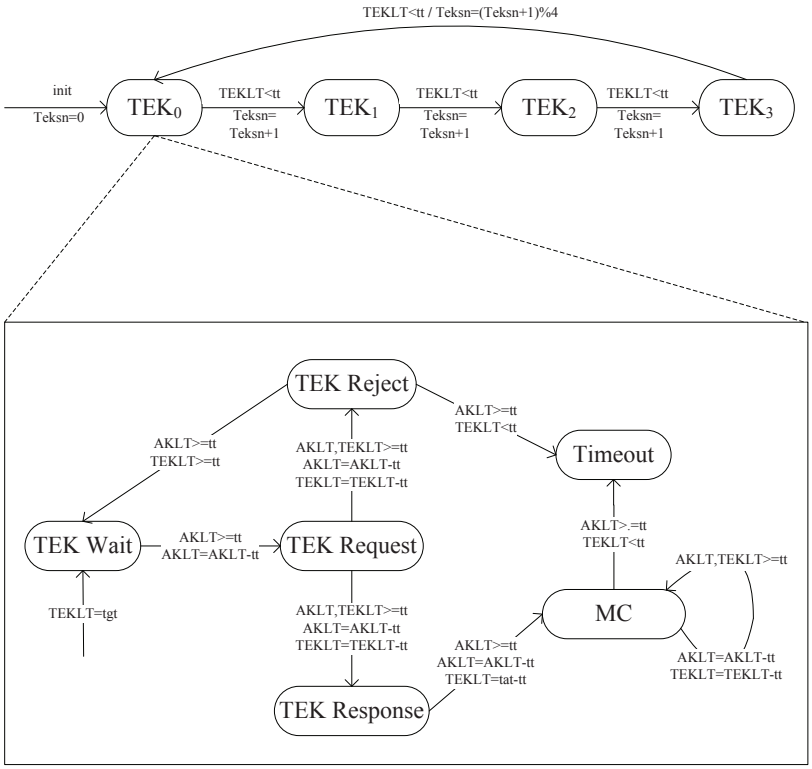

图 4.4: TEK 状态机

SendTEKRequestMsgReal 规则描述了 SS 向 BS 发送 TEK 认证请求的过程。在规则定义的状态转换过程中，if 条件语句声明若 AK 具有大于tt 的生命时长，且 SS 还未生成 TEK 刷新标识，则 SS 生成flag(0) 用于代表首次申请 TEK。TEK 认证请求消息由tekrequest 操作生成，输入参数包括 SAID，PMK_SN 以及 TEK 刷新标识，由cmac 操作生成消息认证码与原文消息一同发送。消息成功发送后，双方连接状态中的AKLT 减少tt 时长，TEK 的请求宽容时间同样减少tt 时长，协议当前执行状态转换为标识密钥传输阶段的ptek。

```
crl [SendTEKRequestMsgReal] :
  (node[A]: nonce(A,B,R0),PMKSN,CMACKEY,SAID,CS1)
  (msgnil) {AKLT,TEKLT,PS}
 =>
  (node[A]: nonce(A,B,R0),PMKSN,CMACKEY,SAID,CS1,flag(0))
  (from A to B send (tekrequest(SAID,PMKSN,flag(0)),
      cmac(tekrequest(SAID,PMKSN,flag(0)),CMACKEY)) )
  {sd(AKLT,tt),sd(tgt,tt),ptek}
```

```
9    if AKLT >= tt /\ not(flag(0) inc CS1) /\ not(flag(1) inc CS1) .
```

RecvTEKResponseMsgReal 规则描述了 SS 接收 BS 发送的密钥响应消息并生成传输密钥 TEK 的过程。发生状态转换前，消息池中存有 BS 向 SS 发送的密钥响应消息MSG8 及其消息验证码。规则中的 if 条件语句声明若从MSG8 中提取出的 SAID 和 PMK_SN 与 SS 本地信息集中相同，且由 MSG8 和 CMAC key 计算出的消息摘要与接收到的消息验证码相同，则 SS 从该密钥响应消息中通过getcontek 和geteks 操作中提取出 TEK 计数器 COUNTER_TEK 和加密密钥序列号 EKS。随后，SS 根据接收到的 COUNTER_TEK 以及密钥协商阶段生成的 CMAC-TEK prekey 和 SAID，通过构造函数tek1 和tek2 生成两组传输密钥。

```
1  crl [RecvTEKResponseMsgReal] :
2    (node[A]: PMKSN,CMACTEK,CMACKEY,SAID,FLAG,CS1)
3    (from B to A send (MSG8,CMAC2)) {AKLT,TEKLT,ptek}
4   =>
5    (node[A]: PMKSN,CMACTEK,CMACKEY,SAID,FLAG,CS1,getcontek(MSG8),geteks(MSG8),
6          tek1(CMACTEK,SAID,getcontek(MSG8),algo),
7          tek2(CMACTEK,SAID,nextc(getcontek(MSG8)),algo) )
8    (msgnil) {AKLT,tat,ptek}
9    if getsaid8(MSG8) == SAID /\ getpmksn8(MSG8) == PMKSN /\
10     equalm(cmac(MSG8,CMACKEY),CMAC2) /\ FLAG == flag(0) .
```

消息传输及重认证通讯过程

在 BS 成功为 SS 维护两组 TEK 后，双方通过生成的传输密钥 TEK 实施加密消息的传输。MessageTransmit 规则描述了在当前 AK 和 TEK 的有效生命周期内，SS 与 BS 通过传输密钥 TEK 加密通信内容，接收方利用相同的 TEK 进行解密。在该规则中，变量TEK1 和TEK2 代表双方维护的两组传输密钥，CS1 和CS2 表示 SS 与 BS 消息集中其它所有收集到的信息。构造函数msgs 用于表示 SS 与 BS 通信的消息内容，并通过当前 TEK 加密后发送。消息发送后连接状态中的通信阶段转换为pmt 所表示的加密消息传输阶段。

```
1  crl [MessageTransmit] :
2    (node[A]: TEK1,TEK2,CS1) (node[B]: TEK1,TEK2,CS2)
3    {AKLT,TEKLT,PS}
4   =>
5    (node[A]: TEK1,TEK2,CS1) (node[B]: TEK1,TEK2,CS2)
6    (msgs) {sd(AKLT,tt),sd(TEKLT,tt),pmt}
```

```
    if AKLT >= tt /\ TEKLT >= tt .
op msgs : -> Message [ctor] .
```

CircleTEKMsgReal 规则描述了在加密消息传输阶段，当前 TEK 生命时长即将耗尽，从而激活重认证 TEK 的过程。在该规则中，if 条件语句代表在 AK 剩余生命时长大于消息传输时间tt，而 TEK 剩余时长小于tt 的情况下，SS 与 BS 删除各自消息集中即将失效的第一组 TEK 的相关信息，包括变量FLAG 表示的 TEK 刷新标识以及失效 TEK，并将第一组 TEK 替换为原来的第二组激活 TEK，便于生成新的第二组 TEK 相关信息。在状态转换后的 BS 信息集中，nextc(C_TEK) 表示 TEK 计数器由原来的数值递增，用于后续生成新的激活 TEK。同时，双方连接状态中的当前通信阶段转换为ptek 表示的密钥传输执行阶段。

```
crl [CircleTEKMsgReal] :
   (node[A]: CMACTEK,SAID,FLAG,TEK1,TEK2,C_TEK,CS1)
   (node[B]: CMACTEK,SAID,FLAG,TEK1,TEK2,C_TEK,CS2)
   {AKLT,TEKLT,PS} (msgnil)
  =>
   (node[A]: CMACTEK,SAID,tek1(CMACTEK,SAID,C_TEK,algo),CS1)
   (node[B]: CMACTEK,SAID,tek1(CMACTEK,SAID,C_TEK,algo),nextc(C_TEK),CS2 )
   (msgnil) {AKLT,tgt,ptek}
   if AKLT >= tt /\ TEKLT < tt .
```

CircleAKMsgReal 规则描述了在 PKMv3 任意阶段，当前 AK 即将耗尽生命周期，系统需要激活 AK 重认证的过程。在本规则中，if 条件语句代表 AK 的剩余生命时间小于传输时间tt 时，SS 和 BS 将本轮密钥分发过程中的密钥信息如 AK、TEK 相关信息、以及安全组件等信息全部删除，仅将各自当前的随机数和各自的数字证书信息保留，便于双方重新进行认证请求。同时，将系统状态中的当前阶段标识转换为pak 表示的双向认证执行阶段。

```
crl [CircleAKMsgReal] :
   (node[A]: R0,CS1) (node[B]: R1,CS2)
   {AKLT,TEKLT,PS} (msgnil)
  =>
   (node[A]: R0,cert(A,macaddr(A),pubkey(A)))
   (node[B]: R1,cert(A,macaddr(A),pubkey(A)))
   {agt,tgt,pak} (msgnil) if AKLT < tt .
```

4.3 入侵者模型

4.3.1 攻击者类型定义

入侵者属于网络环境中的一部分，它可以参与到正常的协议过程中，拥有诚实主体的一切行为能力。此外，入侵者对网络具有完全的控制力，可窃听拦截系统中的任何消息，并利用该消息为自己增加新知识，同时还具有重放已知消息或者篡改消息内容的能力。入侵者通过以中间人的身份参与到诚实主体间的会话当中，或实施穿插攻击或 DoS 攻击，尝试阻止 SS 与 BS 间建立正常通信。本文将网络中遭遇的攻击模拟为同一个入侵者的行为。

在 Maude 中将入侵者定义为`Intruder`类型，并指定它为站点类型`Station`的子类，从而使其能够继承普通站点的所有能力。`Intruder`由输入参数为`String`类型的构造函数`intru`生成。操作符`seed0`代表入侵者的初始随机数。描述入侵者行为的重写规则中出现变量的类型及其具体描述如表所示，变量C 用于代表网络中所出现的入侵者。由于入侵者同样是网络中的主体，其自身拥有 CA 认证中心授予的 X.509 数字证书。同时为了简化程序代码，将规则中非首次出现的入侵者证书定义为`CertC` 类型，并通过变量`Cert_C` 替代。

```
sorts Intruder CertC . subsort CertC < Cert .
subsort Intruder < Station .
op intru : String -> Intruder [ctor] .
op seed0 : -> Rand [ctor].
var C : Intruder . var CERT_C : CertC .
mb cert(C,macaddr(C),pubkey(C))) : CertC .
```

4.3.2 攻击者行为建模

对于双向认证通信阶段消息的每一次收发过程，入侵者均可以选择实施截获，重放和篡改的操作，从而使其以中间人的身份参与到 SS 与 BS 的会话中。

入侵者拦截网络中消息的行为，使用`RecvAuthResponseMsgFake` 规则为例描述。该规则定义了入侵者从消息池中截获认证响应消息的过程。规则中的 if 条件语句表示若能通过 BS 的公钥将响应消息`MSG2` 数字签名的解密，并且从`MSG2` 中提取的 SS 随机值与入侵者本地信息集收集到的随机值相同，则从消息

池中将该条消息取出使得目标 BS 无法接收到响应消息。入侵者将消息截获后，通过getcert2，getnon22 操作提取出 BS 的数字证书和随机值，并尝试使用自己的私钥对getcipher 操作提取出的 PMK 密文进行解密，以及通过getpmksn2 操作提取出 PMK_SN。状态转换后消息池中的消息为空。

```
crl [RecvAuthRequestMsgFake] :
  (node[C]: R2,CERT_C)
  (from A to B send sencrypt( MSG1,PRI1) )
 =>
  (node[C]: R2,CERT_C, getcert1(MSG1),getnon1(MSG1)) (msgnil)
  if getpub(getcert1(sdecrypt(sencrypt(MSG1,PRI1),pubkey(A)))) == pubkey(A) .
```

以SendAuthResponseMsgFake 规则为例描述入侵者实施消息重放的行为。匹配前的系统状态表示入侵者通过截获一系列认证消息，不断收集 SS 与 BS 会话的相关信息，如 SS 和 BS 的数字证书以及它们生成的随机值等。入侵者截获认证响应消息后，可以选择将自己的身份伪造为 BS，并把认证消息原文重新发送给 SS，以中间人的身份参与到 SS 与 BS 的会话中。If 条件语句声明若当前 AK 仍处于有效状态，则入侵者向 SS 发送认证响应消息。消息成功发送后，AK 的剩余生命值减少消息传输时间。入侵者实施中间人攻击的过程，也更大程度上加速了 AK 生命周期的衰减。

```
crl [SendAuthResponseMsgFake] :
  (node[A]: next(R0),CERT_A,nonce(A,B,R0))
  (node[C]: R2,CERT_C,CERT_A,CERT_B,nonce(A,B,R0),nonce(B,A,R1),NC1,PMKSN)
  (msgnil) {AKLT,TEKLT,PS}
 =>
  (node[A]: next(R0),CERT_A,nonce(A,B,R0))
  (node[C]: R2,CERT_C,CERT_A,CERT_B,nonce(A,B,R0),nonce(B,A,R1),NC1,PMKSN)
  (from B to A send sencrypt(authresponse(CERT_B,nonce(A,B,R0),nonce(B,A,R1),
      encrypt(pmk(A,B,nonce(A,B,R0),nonce(B,A,R1),algo),pubkey(A)),PMKSN),
         prikey(B)))
  {sd(AKLT,tt),TEKLT,PS} if AKLT > tt .
```

重写规则SendAuthConfirmMsgChange 描述了入侵者向 BS 发送伪造的认证确认消息的过程。该规则左边的状态描述了入侵者已截获 SS 发送给 BS 的认证确认消息，并从中获取 SS 的物理地址ssaddr(A)。规则中的 if 条件语句表明若此时 BS 没有收到认证确认消息并从中获取 SS 物理地址，且 AK 生命值大于tt 时，入侵者将认证响应消息中 SS 物理地址替换为ssaddr(C)，并通过checksum 操

作生成新的完整性检验值，进而伪装为 SS 将生成的假消息发送给 BS。消息成功发送后，AK 的剩余生命值减少传输时间tt。

```
crl [SendAuthConfirmMsgChange] :
  (node[B]: PMKSN,AKCOUNT,CS1 )
  (node[C]: nonce(B,A,R1),PMKSN,ssaddr(A),CS2)
  (msgnil) {AKLT,TEKLT,pak}
 =>
  (node[B]: PMKSN,AKCOUNT,CS1 )
  (node[C]: nonce(B,A,R1),PMKSN,ssaddr(A),CS2 )
  (from A to B send (authconfirm(nonce(B,A,R1),ssaddr(C)),
      checksum(authconfirm(nonce(B,A,R1),ssaddr(C)),icv)))
  {sd(AKLT,tt),TEKLT,pak}
  if AKLT >= tt /\ not(ssaddr(A) inc CS1) /\ not(ssaddr(C) inc CS1) .
```

由于安全组件协商阶段以及传输密钥生成阶段中传输的每条消息都通过CMAC消息验证码保护，使得接收者可以验证消息的认证性和完整性。在这两个阶段，入侵者伪造的消息总是会被接收者识破，因此通过重写规则描述入侵者截获相应消息并进行重放的过程。

以ReplayThreeWayConfirmMsg 消息为例描述入侵者在安全组件协商阶段实施消息重放的具体操作。该重写规则定义了入侵者截获 BS 发送的三次握手确认消息，并将消息原文重放发送给 SS 的过程。在规则实施状态转换前，消息池中存在 BS 发送的三次握手确认消息MSG6 及其消息验证码CMAC1。If 条件语句声明若从MSG6 提取出的 PMK_SN 和 SS 随机值与入侵者在之前 SS 与 BS 的会话中搜集到的 PMK_SN 和 SS 随机值相同，入侵者便从消息池中窃取出MSG6，通过getsaid6 操作获取消息中的 SAID，同时将消息原文以 BS 的名义发送给目标 SS，并使得 AK 的剩余生命时间减少tt。

```
crl [ReplayThreeWayConfirm] :
  (node[C]: PMKSN,nonce(A,B,R0),SNP,SECU,CS2)
  (from B to A send (MSG6,CMAC1)) {AKLT,TEKLT,psa}
 =>
  (node[C]: PMKSN,nonce(A,B,R0),SNP,SECU,CS2,getsaid6(MSG6))
  (from B to A send (MSG6,CMAC1)) {AKLT,TEKLT,psa}
  if getpmksn6(MSG6) == PMKSN /\ equals(getnon6(MSG6),nonce(A,B,R0))
    /\ AKLT >= tt /\ TEKLT >= tt .
```

以ReplayTEKResponseMsg 规则为例描述传输密钥生成阶段入侵者实施重放攻击的行为。该规则定义了入侵者从消息池中截获 BS 发送的密钥响应消息，

并伪造为 BS 将消息原文发送给 SS 的过程。在该规则中，变量MSG8 代表密钥响应消息，其消息验证码为CMAC2。If 条件语句定义若从MSG8 中提取的 SAID 和 PMK_SN 的值与入侵者收集的信息集中的相一致，并且密钥 AK 和 TEK 都拥有足够的生命时长，入侵者从消息池中截获MSG8，通过getcontek 操作和geteks 操作从中提取出 COUNTER_TEK 和 EKS，并将消息原文以 BS 的名义发送给 SS。若入侵者在之前阶段已获取 CMAC-TEK prekey，则可通过tek1 操作生成 TEK，否则无法生成传输密钥 TEK 用于破解 SS 与 BS 后续的加密通信内容。

```
crl [ReplayTEKResponseMsg] :
  (node[C]: PMKSN,SAID,FLAG,CS1)
  (from B to A send (MSG8,CMAC2)) {AKLT,TEKLT,ptek}
 =>
  (node[C]: PMKSN,SAID,FLAG,CS1,getcontek(MSG8),geteks(MSG8),
      if (CMACTEK inc CS1) then tek1(CMACTEK,SAID,getcontek(MSG8),algo)
          else contnil fi)
  (from B to A send (MSG8,CMAC2)) {sd(AKLT,tt),sd(TEKLT,tt),ptek}
  if getsaid8(MSG8) == SAID /\ getpmksn8(MSG8) == PMKSN
    /\ AKLT >= tt /\ TEKLT >= tt .
```

4.4　本章小结

本章提出了通过形式化规范语言 Maude 实现对 PKMv3 协议建模的方法。指出将 PKMv3 协议环境视为一个特定系统，通过函数模块定义协议相关数据类型，如站点、证书、消息以及密钥等，并利用等式定义各数据类型的操作，从而构造出系统的静态状态组成。系统的动态行为如站点间消息的收发、连接状态的变化通过系统模块中的重写规则定义。本文所定义的重写规则实现对协议各执行阶段、重认证过程以及入侵者攻击行为的描述，能够真实模拟 PKMv3 协议过程。

第五章　PKMv3 协议的形式化验证

本文利用 Maude 语言实现对 PKMv3 协议的形式化建模。通过函数模块实现协议中站点与消息相关数据类型的形式化定义，描述站点的基本属性，消息保护方式，各密钥生成算法以及具体消息组成。通过系统模块中的重写规则描述站点间消息传输过程，实现 PKMv3 协议各执行阶段，以及重认证过程的建模。同时，在系统模块中通过重写规则描述入侵者实施截获，重放或伪造消息的行为。函数模块和系统模块中定义的协议规范构成 PKMv3 协议模型。基于已建立的协议模型，定义系统初始状态为模型构造实例，通过模型检测的验证方式实现对 PKMv3 协议时间特性以及安全特性的验证。时间特性代表协议模型能够准确模拟协议各阶段的执行过程及执行顺序，可通过 Maude 提供的 LTL 模型检测工具验证；安全特性代表协议设计所满足的安全性质，通过 Maude 提供的 search 语句穷尽系统执行路径，检测在任意状态下均能满足的安全性质。PKMv3 的验证流程如图 5.1 所示。

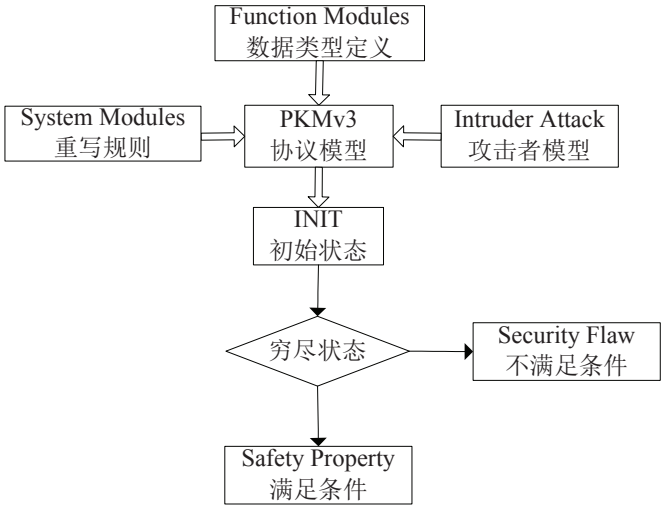

图 5.1: PKMv3 验证流程图

5.1 协议初始状态的定义

PKMv3 协议所处的网络环境初始状态通过`init` 操作指定，`init` 是系统状态 States 类型的常量。通过`eq` 关键字定义初始状态中存在名为`a` 的手机站、名为`b` 的基站以及名为`c` 的入侵者，它们拥有各自的数字证书以及初始随机数。此时网络消息池的消息为空，用操作符`msgnil` 表示。基站与手机站间的连接状态各变量均设置为初始数值，AK 和 TEK 剩余生命时长为 AK 宽容时间以及 TEK 宽容时间，当前协议执行状态为`pak` 表示的双向认证过程。根据 IEEE802.16 标准，将 AK 宽容时间`agt` 与预定义生命时长`aat` 的测试数值分别定义为 60 和 300，将 TEK 宽容时间`tgt` 和预定义生命时长`tat` 的测试数值分别定义为 60 和 180，并将 SS 与 BS 间消息传输时间定义为定长 4。

```
op init : -> States .
eq init =
  (node[ss("a")]: seed,cert(ss("a"),macaddr(ss("a")),pubkey(ss("a"))))
  (node[bs("b")]: seed1,cert(bs("b"),macaddr(bs("b")),pubkey(bs("b"))))
  (node[intru("c")]: seed0,cert(intru("c"),macaddr(intru("c")),pubkey(intru("c"))))
  (msgnil) {agt,tgt,pak} .
eq agt = 60 . eq aat = 300 .
eq tgt = 60 . eq tat = 180 . eq tt = 4 .
```

5.2 PKMv3 协议验证过程

5.2.1 时间特性的验证

Maude 提供的 LTL 模型检测工具执行存在前提条件，即从定义为 State 类型的初始状态开始，规定执行系统中所有匹配的重写逻辑后的可达状态为有限个数。系统中定义的重写规则必须满足执行后的可达状态均为 State 类型。将协议中特定的系统状态定义为原子命题`Prop`，`Prop` 类型为 LTL 公式 Formula 的子类型，即原子命题可以通过基本操作符 $\rightarrow$，U，W，`<>`，`[]`，$\sim$ 生成 LTL 公式。最终 Maude 从初始状态开始，验证协议模型是否满足各 LTL 公式定义的时间特性。

定义 PKMv3 模型中的系统状态类型`States` 为`State` 类型的子类。其次定义用于表示数据类型为 Prop 的原子命题的操作符，并通过 $\models$ 操作符实现各原子命

题的等式定义。每个原子命题与均与某个特定的预测系统状态相匹配，其中eap-auth 代表协议当前执行到双向认证阶段，nego-sa 代表处于安全组件认证阶段，exch-tek 代表处于传输密钥生成阶段，msg-trans 代表处于加密消息传输阶段，inva-msg 代表消息传输中使用失效的认证密钥或传输密钥，req-ak 代表 AK 重认证过程，none-tek 代表当前 TEK 为失效状态或为空。

```
subsort States < State .
ops eap-auth nego-sa exch-tek msg-trans inva-msg
     req-ak none-tek : -> Prop .
var S1 : States . var C1 : Contents .
var AK : Ak . var TEK : Tek .
vars AKLT TEKLT : LTime . var PS : Process .
eq (S1 {AKLT,TEKLT,pak}) |= eap-auth = true .
eq (S1 {AKLT,TEKLT,psa}) |= nego-sa = true .
eq (S1 {AKLT,TEKLT,ptek}) |= exch-tek = true .
eq (S1 {AKLT,TEKLT,pmt}) |= msg-trans = true .
ceq (S1 {AKLT,TEKLT,PS}) |= inva-msg = true
   if AKLT < zt or TEKLT < zt .
eq (S1 {agt,TEKLT,pak}) |= req-ak = true .
eq (S1 {AKLT,tgt,pak}) |= none-tek = true .
```

连续性

PKMv3 协议规范定义从初始化过程开始，SS 间与 BS 依次执行双向认证，安全组件协商，密钥生成，加密消息传输的过程，为了验证实验模型的准确性，通过以下red 命令实现 PKMv3 协议连续性（Succession）的验证。

```
Maude> red modelCheck(init,(eap-auth U nego-sa) /\ (<> nego-sa U exch-tek) /\ (<>
     exch-tek U msg-trans)) .

reduce in CHECKS : modelCheck(init, (eap-auth U nego-sa) /\ (<> nego-sa U exch-
    tek) /\ (<> exch-tek U msg-trans)) .
rewrites: 457 in 4ms cpu (2ms real) (114250 rewrites/second)
result Bool: true
```

命令中的 LTL 公式表述从初始状态开始，执行双向认证阶段直到安全组件协商阶段，执行安全组件协商阶段直到传输密钥生成阶段，执行传输密钥生成直到加密消息传输阶段。实验结果返回为 true，表明 PKMv3 协议模型从初始状态开始，能够成功通过匹配描述消息传输的重写规则，依次执行各协议阶段。

密钥活性

在协议执行的各个阶段，认证密钥以及传输密钥应时刻保持生命活性。通过以下指令实现 PKMv3 协议执行过程中密钥活性（Key Liveness）的验证。

```
Maude> red modelCheck(init,[]~ inva-msg ) .

reduce in CHECKS : modelCheck(init, []~ inva-msg) .
rewrites: 1527 in 4ms cpu (4ms real) (381750 rewrites/second)
result Bool: true
```

该命令中的 LTL 公式表明从初始状态开始可达的任意阶段，协议中出现任意失效 AK 或 TEK 的情况都是不允许的。执行结果返回布尔值 true，表明 PKMv3 协议模型的执行过程中总能保持密钥的频繁刷新，保障消息传输过程中密钥生命时长始终大于 0。

AK 生命周期

认证密钥 AK 具有生命周期，当前 AK 生命时长耗尽后需要执行重认证过程。通过以下指令检测在加密消息传输阶段 AK 失效后，系统是否能够重新请求到新的激活 AK，保证新旧 AK 生命周期（Period of AK）的持续更替。

```
Maude> red modelCheck (init ,(<> msg-trans U eap-auth U nego-sa U exch-tek U msg-
    trans )) .

reduce in CHECKS : modelCheck(init, ((<> msg-trans U eap-auth) U nego-sa) U (exch
    -tek U msg-trans)) .
rewrites: 378 in 4ms cpu (2ms real) (94500 rewrites/second)
result Bool: true
```

本条命令中的 LTL 公式表述存在一条协议路径，依次实现消息传输，双向认证，安全组件协商，传输密钥生成过程，最终又回到加密消息传输的过程。执行结果返回为 true，表明当 AK 即将失效时，系统能够自动匹配 AK 重认证规则，实现新一轮 AK 生命迭代。

TEK 生命周期

传输密钥 TEK 同样具有生命周期，且它的生命周期嵌入在 AK 的生命周期中。通过如下所示的指令实现 TEK 重认证，以及 AK 失效后自动丢弃相关 TEK 等 TEK

生命周期（Period of TEK）性质的检测。

```
Maude> red modelCheck (init ,(<> msg-trans U exch-tek U msg-trans) /\ ( [] req-ak
     -> none-tek) ) .

reduce in CHECKS : modelCheck(init, ((<> msg-trans U exch-tek) U msg-trans) /\
    ([]req-ak -> none-tek)) .
rewrites: 412 in 4ms cpu (2ms real) (103000 rewrites/second)
result Bool: true
```

本条命令中的 LTL 公式表述存在一条协议路径，依次实现消息传输，传输密钥生成，消息传输的过程，同时满足 AK 重认证过程中当前 TEK 为失效状态。执行结果显示为 true，表明协议模型能够正确模拟 TEK 真实生命周期的迭代过程。

5.2.2　安全特性的验证

基于已定义的网络环境的状态模型以及描述站点行为状态迁移的重写规则，可实现 PKMv3 协议模型的仿真与验证。通过穷尽搜索协议状态空间判断某些特定状态的可达性，从而检验出协议的安全性质。使用 Maude 中的`search` 指令实现 PKMv3 协议模型的安全特性检测，该指令以初始状态为起点，遍历系统的全部重写行为，并找出所有与规定模式相匹配的状态以及检验协议相应的安全性质。

机密性

协议的机密性（Confidentiality）表现为在诚实主体的通信过程中，入侵者无法获取协议执行过程中交换或生成的密钥信息。PKMv3 协议的目标是最终实现传输密钥的生成，因此保证传输密钥不被泄露对协议的安全通信有着十分重要的意义。使用如下`search` 指令以初始状态为起点，检测所有可达的最终状态，查找侵入者的消息集中存在传输密钥的可达路径。

```
Maude> search init =>* (S:States) (node[intru("c")]: C:Contents, tek1(cmactek(
    ak(pmk(ss("a"), bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),
    seed1),algo),bs("b"),ssaddr(ss("a")), algo),akcount(nonce(ss("a"),bs("b"),seed
    ),nonce(bs("b"),ss("a"),seed1)),algo),said(nonce(bs("b"), ss("a"),next(seed1))
    ,nonce(ss("a"),bs("b"),next(seed)),secucapa(ss("a"))),countertek(nonce(ss("a")
    ,bs("b"),next(seed)),nonce(bs("b"),ss("a"),next(seed1))),algo) ) .

search in INTRUDER : init =>* S:States node[intru("c")]: C:Contents,tek1(
    cmactek(ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed), nonce(bs("b"),ss("
    a"),seed1),algo), bs("b"),ssaddr(ss("a")),algo), akcount(nonce(ss("a"),bs("b")
```

```
    ,seed), nonce(bs("b"),ss("a"),seed1)),algo),said(nonce(bs("b"),ss("a"),next(
    seed1)),nonce(ss("a"),bs("b"),next(seed)),secucapa(ss("a"))),countertek(nonce(
    ss("a"),bs("b"),next(seed)),nonce(bs("b"),ss("a"),next(seed1))),algo) .

No solution.
states: 539 rewrites: 13206 in 28ms cpu (60ms real) (471642 rewrites/second)
```

在该查找指令中，传输密钥由操作符tek1 构造生成，变量C 代表入侵者消息集中的其它信息。执行该指令返回结果为不存在解决方案，表明侵入者在任意可达状态下都不可能窃取相关密钥材料并生成传输密钥 TEK。因此，该协议能够满足密钥的机密性，保证加密消息的安全传输。

认证性

认证性（Authentication）是密钥管理协议最重要的安全性质，协议的认证性体现在即使存在入侵者的参与，SS 和 BS 间仍能相互确认真实身份，同时入侵者无法实现身份伪造，并无法搜集到关键材料生成认证密钥。执行search 语句查找协议所有的系统状态中入侵者信息集中存在认证密钥 AK 的情况。

```
search init =>* (S:States) (node[intru("c")]: C:Contents, ak(pmk(ss("a"),bs("b"))
    ,nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),algo),bs("b"),ssaddr
    (ss("a")),algo)) .

search in INTRUDER : init =>* S:States node[intru("c")]: C:Contents,ak(pmk(ss("a
    "),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),algo),bs("
    b"),ssaddr(ss("a")),algo) .

No solution.
states: 539 rewrites: 13206 in 20ms cpu (53ms real) (660300 rewrites/second)
```

在该查找语句中，变量S 用于表示系统中入侵者结点之外的其它状态组成，变量C 则表示入侵者消息集中除 AK 外的其它消息。执行结果表明，入侵者总是无法获取到认证密钥，从而无法作为正常通信主体参与到站点间的建立的安全连接会话中去。由此，该密钥管理协议能够保证站点间认证的可靠性。

完整性

协议的完整性（Integrity）保证是指诚实主体间传递的消息不能被误传和篡改，或至少消息接收方能够识别出错误的消息数据。在消息的请求和响应阶段均使用了

数字签名保障完整性和认证性，同时在三次握手以及密钥传输阶段均采用 CMAC 摘要实施消息认证性和完整性保护，因此主要检测认证确认消息传输的完整性，使用search 语句检测 BS 接收到错误的 SS 地址，从而生成与 BS 不一致的 AK 的可能性。该查找语句指定最多返回一条满足条件的重写路径。

```
Maude> search [1] init =>* S:States node[bs("b")]: C:Contents, A:Ssaddr, K:Ak
    such that A:Ssaddr =/= ssaddr(ss("a")) = true .

search in INTRUDER : init =>* S:States node[bs("b")]: C:Contents,A:Ssaddr,K:Ak
    such that A:Ssaddr =/= ssaddr(ss("a")) = true .

Solution 1 (state 34)
states: 35 rewrites: 580 in 12ms cpu (9ms real) (48333 rewrites/second)

S:States --> msgnil {80,60,pak}
(node[ss("a")]: next(seed), ssaddr(ss("a")),
pmksn(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1
    ),algo)),
akcount(nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1)),
nonce(ss("a"),bs("b"),seed), nonce(bs("b"), ss("a"),seed1),
cert(ss("a"),macaddr(ss("a")),pubkey(ss("a"))),
cert(bs("b"),macaddr(bs("b")),pubkey(bs("b"))),
ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),
    algo),bs("b"),ssaddr(ss("a")),algo),
pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),algo
    ))
(node[intru("c")]: seed0, ssaddr(ss("a")),
pmksn(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1
    ),algo)),
decrypt(encrypt(pmk(ss("a"), bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss
    ("a"),seed1), algo),pubkey(ss("a"))),prikey(intru("c"))),
nonce(ss("a"),bs("b"),seed), nonce(bs("b"),ss("a"),seed1),
cert(ss("a"),macaddr(ss("a")),pubkey(ss("a"))),
cert(bs("b"),macaddr(bs("b")),pubkey(bs("b"))),
cert(intru("c"), macaddr(intru("c")),pubkey(intru("c"))))
C:Contents --> next(seed1),
pmksn(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1
    ),algo)),
akcount(nonce(ss("a"),bs("b"),seed), nonce(bs("b"),ss("a"),seed1)),
nonce(ss("a"),bs("b"),seed), nonce(bs("b"), ss("a"),seed1),
cert(ss("a"),macaddr(ss("a")),pubkey(ss("a"))),
cert(bs("b"),macaddr(bs("b")),pubkey(bs("b"))),
pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),algo
    )
A:Ssaddr --> ssaddr(intru("c"))
K:Ak --> ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a")
    ,seed1),algo),bs("b"),ssaddr(intru("c")),algo)
```

状态查找结果返回一条重写路径，系统状态中变量A 表示 BS 信息集中的 SS 地址信息，可看出 BS 接收到入侵者 c 篡改过的错误 SS 地址ssaddr(intru("c"))，

并用其生成了无效的认证密钥 AK，使得协议的完整性遭到破坏。同时，查看变量S中入侵者信息集的内容，得出入侵者能够窃取 SS 与 BS 数字证书等信息。由于认证阶段的第三条确认消息使用了默认的消息校验和算法来防止数据包在网络传输中的误传，入侵者可以篡改消息中的数据，并自行生成修改后信息的校验和附在消息末尾。接收到消息的 BS 只能检测消息在物理网络传输中的准确性，并无法保障消息的完整性。一种可行的改进方案是使用 AK 计算出认证确认消息的校验和，BS 通过接收到的 SS 地址生成 AK，再将 AK 与收到的明文消息计算的结果和校验和对比是否一致。因为入侵者始终都无法获得 PMK 的值，从而无法生成 AK 和消息校验和，而使用 AK 作为算子生成校验和则能保障消息不被非法用户篡改。

可用性

协议执行过程中应时刻保证系统计算资源以及密钥的可用性（Availability），计算资源通常由于遭受网络中的 DoS 攻击以至可用性无法保障，而密钥的可用性则意味着诚实主体间的正常通信不能遭受中间人攻击。由于消息的每一次发送过程都会损耗密钥的生命时长，因此入侵者消息重放的行为会导致在相同密钥生命周期内，用于正常通信的时间大大缩短。使用以下`search` 语句查找入侵者搜集 SAID 等密钥消息所执行的重写路径。

```
Maude> search [1] init =>* (S:States) (node[intru("c")]: C:Contents, said(nonce(
    bs("b"),ss("a"),next(seed1)),nonce(ss("a"),bs("b"),next(seed)),secucapa(ss("a
    "))) ) .

search in INTRUDER : init =>* S:States node[intru("c")]: C:Contents,said(nonce(
    bs("b"),ss("a"),next(seed1)),nonce(ss("a"),bs("b"),next(seed)),secucapa(ss("a
    "))) .

Solution 1 (state 77)
states: 78 rewrites: 1486 in 0ms cpu (4ms real) (~ rewrites/second)

S:States --> {68,60,psa}
(node[ss("a")]: next(next(seed)), secucapa(ss("a")), snp(64, 1000),
pmksn(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1
    ),algo)),
akcount(nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1)),
cmackey(cmactek(ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),
    ss("a"),seed1),algo),bs("b"),ssaddr(ss("a")),algo),akcount(nonce(ss("a"),bs("b
    "),seed),nonce(bs("b"),ss("a"),seed1)),algo),algo),
nonce(ss("a"),bs("b"),next(seed)),nonce(bs("b"),ss("a"),next(seed1)),
```

```
14 cmactek(ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),
       seed1),algo),bs("b"),ssaddr(ss("a")), algo),akcount(nonce(ss("a"),bs("b"),seed
       ),nonce(bs("b"),ss("a"),seed1)),algo),
15 ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),
       algo),bs("b"),ssaddr(ss("a")),algo),
16 akid(ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),
       seed1),algo),bs("b"),ssaddr(ss("a")),algo),pmksn(pmk(ss("a"),bs("b"),nonce(ss
       ("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),algo)),ss("a"),bs("b"),algo))
17 (node[bs("b")]: eks, next(next(seed1)), secucapa(ss("a")), snp(64,1000),
18 pmksn(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1
       ),algo)),
19 akcount(nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1)),
20 cmackey(cmactek(ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),
       ss("a"),seed1), algo),bs("b"),ssaddr(ss("a")),algo),akcount(nonce(ss("a"),bs("
       b"),seed),nonce(bs("b"),ss("a"),seed1)),algo),algo),
21 nonce(ss("a"),bs("b"),next(seed)), nonce(bs("b"),ss("a"),next(seed1)),
22 cmactek(ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),
       seed1),algo),bs("b"),ssaddr(ss("a")),algo),akcount(nonce(ss("a"),bs("b"),seed)
       ,nonce(bs("b"),ss("a"),seed1)),algo),
23 said(nonce(bs("b"),ss("a"),next(seed1)),nonce(ss("a"),bs("b"),next(seed)),
       secucapa(ss("a"))),
24 ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),
       algo),bs("b"),ssaddr(ss("a")),algo),
25 akid(ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),
       seed1),algo),bs("b"),ssaddr(ss("a")), algo),pmksn(pmk(ss("a"),bs("b"),nonce(ss
       ("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),algo)),ss("a"),bs("b"),algo))
26 (from bs("b") to ss("a") send cmac(threewayconfirm(nonce(ss("a"),bs("b"),next(
       seed)),pmksn(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss
       ("a"),seed1),algo)),snp(64,1000),said(nonce(bs("b"),ss("a"),next(seed1)),nonce
       (ss("a"),bs("b"),next(seed)),secucapa(ss("a")))),cmackey(cmactek(ak(pmk(ss("a
       "),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"), seed1),algo),bs
       ("b"),ssaddr(ss("a")),algo),akcount(nonce(ss("a"),bs("b"),seed),nonce(bs("b"),
       ss("a"),seed1)),algo),algo)),
27 threewayconfirm(nonce(ss("a"),bs("b"),next(seed)),pmksn(pmk(ss("a"),bs("b"),nonce
       (ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),algo)),snp(64,1000),said(
       nonce(bs("b"),ss("a"),next(seed1)),nonce(ss("a"),bs("b"),next(seed)),secucapa(
       ss("a")))))
28 C:Contents --> seed0, ssaddr(ss("a")), secucapa(ss("a")), snp(64, 1000),
29 pmksn(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed), nonce(bs("b"),ss("a"),
       seed1),algo)),
30 decrypt(encrypt(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss
       ("a"),seed1),algo),pubkey(ss("a"))), prikey(intru("c"))),
31 nonce(ss("a"),bs("b"),seed), nonce(ss("a"),bs("b"),next(seed)),
32 nonce(bs("b"),ss("a"),seed1), nonce(bs("b"),ss("a"),next(seed1)),
33 cert(ss("a"),macaddr(ss("a")),pubkey(ss("a"))),
34 cert(bs("b"),macaddr(bs("b")),pubkey(bs("b"))),
35 cert(intru("c"),macaddr(intru("c")),pubkey(intru("c"))),
36 akid(ak(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),
       seed1),algo), bs("b"),ssaddr(ss("a")),algo),pmksn(pmk(ss("a"),bs("b"),nonce(ss
       ("a"),bs("b"),seed),nonce(bs("b"), ss("a"),seed1),algo)),ss("a"),bs("b"),algo)
```

返回其中一条满足条件的执行路径，状态编号为 77，并通过`show path labels 77` 语句打印出该状态所执行的重写规则名称。由此可以看出，入侵

者以中间人的身份参与在双向认证以及安全组件协商阶段的每一条消息的收发过程中，使得认证密钥的生命时长成倍递增。因此，该协议因遭受中间人攻击无法保证密钥的可用性，可通过在消息传输中加入时间戳从而预防重放消息。同时对于无线网络系统来说，CMAC 消息验证码的生成需要较大的计算量，因此攻击者可通过大量发送错误三次握手请求消息，造成 BS 计算资源的占用，实现 DoS 攻击。

```
Maude> show path labels 77 .
SendAuthRequestMsgReal RecvAuthRequestMsgFake SendAuthRequestMsgFake
RecvAuthRequestMsgReal SendAuthResponseMsgReal RecvAuthResponseMsgFake
SendAuthResponseMsgFake RecvAuthResponseMsgReal SendAuthConfirmMsgReal
RecvAuthConfirmMsgFake SendAuthConfirmMsgFake RecvAuthConfirmMsgReal
SendThreeWayRequestMsgReal ReplayThreeWayRequestMsg RecvThreeWayRequestMsgReal
SendThreeWayResponseReal ReplayThreeWayResponse RecvThreeWayResponseReal
SendThreeWayConfirmReal ReplayThreeWayConfirm
```

5.2.3 验证结果分析

本文针对 PKMv3 协议时间特性与安全特性的验证结果如表 5.1 所示。该表中详细的描述了各时间特性以及安全特性的验证方法、重写次数、执行时间以及验证结果。由此可以得出 PKMv3 协议模型能够满足协议标准中规定的执行阶段连续性，密钥生命时长新鲜性，AK 以及 TEK 的重认证性，并且能够保证协议密钥的机密性以及站点认证的可靠性，却无法满足消息传输完整性以及资源可用性。

表 5.1: 协议性质验证结果

PKMv3	协议特性	验证方法	重写次数	执行时间	验证结果
时间特性	连续性	LTL 模型检测	457	4ms	通过
	密钥活性	LTL 模型检测	1527	4ms	通过
	AK 生命周期	LTL 模型检测	378	4ms	通过
	TEK 生命周期	LTL 模型检测	412	4ms	通过
安全特性	机密性	Search 指令	13206	28ms	通过
	认证性	Search 指令	13206	53ms	通过
	完整性	Search 指令	580	12ms	不通过
	可用性	Search 指令	1486	4ms	不通过

5.3　本章小结

本章通过 Maude 形式化建模工具实现 PKMv3 协议模型时间特性以及安全特性的验证。Maude 描述的 PKMv3 协议规范是可执行的，可以实现协议执行过程的模拟仿真。通过 Maude 提供的 LTL 模型检测工具，对协议模型的连续性，密钥活性，认证密钥以及传输密钥的生命周期进行验证，得出该协议能够满足协议标准中定义的各项时间特性。通过 Maude 中提供的 search 指令，进行协议系统状态空间搜索，得出 PKMv3 协议能够满足认证的可靠性，密钥的机密性等安全性质，但会遭到中间人攻击使得消息传输的完整性，密钥资源的可用性以及数字证书的可用性无法保障，最终对实验验证结果进行分析总结，阐述各协议性质的验证方法、验证时间与验证结果。

第六章　PKMv3 协议的改进

根据 PKMv3 协议模型的验证结果可知，该协议可能会遭受到 DoS 攻击，重放攻击以及中间人攻击使得消息传输的完整性，协议密钥的可用性，以及数字证书的机密性无法得到保障。针对目前 PKMv3 中出现的主流缺陷，本文将原密钥协议中双向认证阶段，安全组件协商阶段以及传输密钥生成阶段中传输的消息内容进行修改，提出了一种安全密钥管理协议，能够有效预防入侵者的各类攻击行为。

6.1　安全密钥管理协议

改进后的密钥管理协议中各协议阶段的通信内容如图 6.1 所示。在双向认证过程中，通过引入 CA 认证中心实现 SS 与 BS 数字证书的请求与验证，从而避免原协议中无法保证数字证书机密性的问题，同时增加认证可靠性。CA 认证中心是电子商务认证授权机构，负责发送和管理数字证书，可作为可信第三方提供数字证书和公钥密钥的合法性检验。针对协议易遭受到中间人攻击，损耗密钥生命时长从而破坏协议可用性的缺陷，通过在 SS 与 BS 间传输的消息中加入时间戳，时间戳代表消息发送的全局时间，有助于消息接收者识别出重放消息并将其丢弃。

此外，通过使用 AK 作为算子生成认证确认消息的校验和，得以保证传输消息的完整性。由于 PKMv3 协议中安全组件协商与密钥生成阶段传输的消息均通过 CMAC 消息验证码保护，系统需要先计算并比较消息验证码的正确性，再进行后续通信。这一特性使得入侵者可以通过向目标站点发送大量无效消息，使其耗尽资源从而无法提供服务。针对这两个阶段遭受的 DoS 攻击，可在消息中通过添加一种叫做 SAI(Shared Authentication Information) 的共享认证信息成功避免。

CA Authentication:
Msg1. SS -> CA: SS
Msg2. CA -> SS: [SS | $ENC_{CA}(Cert_{SS})$]SIG_{CA}(2)
Msg3. SS -> BS: [$ENC_{CA}(Cert_{SS})$ | $ENC_{BS}(N_S$ | Capabilities) | T_2]SIG_{SS}(3)
Msg4. BS -> CA: BS
Msg5. CA -> BS: [BS | $ENC_{CA}(Cert_{BS})$]SIG_{CA}(5)
Msg6. BS -> CA: [$ENC_{CA}(Cert_{SS}$ | $Cert_{BS})$ | N_S | N_B]SIG_{BS}(6)
Msg7. CA -> BS: [$ENC_{CA}(Cert_{BS}$ | N_S | $N_B)$]SIG_{CA}(7)
Msg8. BS -> SS: [$ENC_{CA}(Cert_{BS}$ | N_S | $N_B)$ | ENC_{SS}(PMK | PMK_SN) | T_8]SIG_{BS}(8)
Msg9. SS -> CA: [$ENC_{CA}(Cert_{BS}$ | N_S | N_B)]SIG_{SS}(9)
Msg10. CA -> SS: [$ENC_{SS}(N_S$ | $N_B)$]SIG_{CA}(10)
Msg11. SS -> BS: N_B | SS_Address | T_{11} | AK(N_B | SS_Address | T_{11})
SA Negotiation:
Msg12. BS -> SS: N_B | PMK_SN | AKID | T_{12} | CMAC(12) | SAI
Msg13. SS -> BS: N_B | N_S | PMK_SN | AKID | SNP | SC | T_{13} | CMAC(13) | SAI
Msg14. BS -> SS: N_S | PMK_SN | SAID | SNP | T_{14} | CMAC(14) | SAI
TEK Transmission:
Msg15. SS -> BS: SAID | PMK_SN | TEK Refresh Flag | T_{15} | CMAC(15) | SAI
Msg16a. BS -> SS: SAID | PMK_SN | COUNTER_TEK | EKS | T_{16} | CMAC(16a) | SAI
Msg16b. BS -> SS: SAID | Wrong_Code | T_{17} | CMAC(16b) | SAI
Message Communication:
Msg17. BS <-> SS: EKS | PN | Communication Contents | ICV | TEK(17)

图 6.1: 改进后的密钥管理协议

6.1.1 认证阶段消息传输

在协议通信中，各站点与 CA 的通信路径为安全链路，未实施加密消息传输的 SS 与 BS 间通信链路则可能会遭遇攻击。改进后的密钥管理协议认证阶段中传输的各消息类型如图 6.2 所示。协议中的消息 1 和消息 2 表示 SS 向 CA 请求自身的数字证书，CA 向 SS 返回其身份标识以及通过 CA 公钥加密的 SS 证书，并将整条消息通过数字签名保护。与之相类似，在消息 4 和消息 5 中，BS 向 CA 请求自身的数字证书，CA 向 BS 返回其身份标识以及通过 CA 公钥加密的 SS 证书，并将整条消息通过数字签名保护。消息 3 代表当 SS 获取数字证书后，向 BS 发送认证请求消息，包括由 CA 加密的 SS 数字证书，通过 BS 公钥加密的 SS 随机值以及 SS 加密能力，表明消息发送时间点的时间戳，并将整条消息通过 SS 的私钥进行数字签名。BS 接收到该认证请求消息，则向 CA 发送消息 6 用于验证 SS 证书的合法

性，包括 CA 加密的 SS 与 BS 数字证书，SS 随机值，BS 随机值，并将消息进行数字签名保护。若 SS 通过 CA 身份验证，CA 则向 BS 发送消息 7，包括通过 CA 公钥加密保护以及数字签名保护的 BS 证书和随机值。

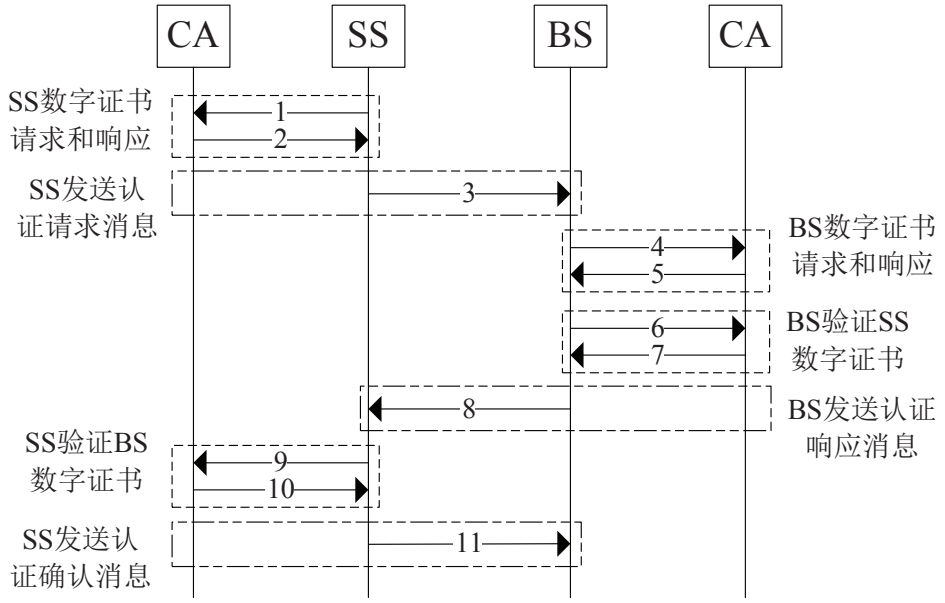

图 6.2: 认证阶段各消息类型

由此，BS 向 SS 发送认证响应消息 8，该消息内容通过数字签名保护，并包括由 CA 加密保护的 BS 证书和随机值，通过 SS 公钥加密保护的 PMK、PMK 序列号，以及时间戳。SS 接收到该认证响应消息，向 CA 发送消息 9 请求验证 BS 数字证书的真实性。若验证成功则 CA 向 SS 发送消息 10，包括用 CA 公钥加密的 SS 随机值与 BS 随机值。SS 比较接收到的随机值与自己生成的随机值是否一致，消息 11 表示若比较结果一致，则 SS 生成 AK 并向 BS 发送认证确认消息，包括 BS 随机值，SS 的物理地址，发送消息时间戳，以及消息校验和，其中消息校验和通过 AK 作为初始算子计算生成。BS 对比随机值以及校验和的正确性，并最终同样生成授权密钥 AK。

6.1.2　共享认证消息传输

安全密钥管理协议中安全组件协商阶段与密钥生成阶段中消息通信过程如图 6.1 中第 12 条至第 16 条消息所示。与原 PKMv3 协议相比，每一条由 CMAC 消息

验证码消息内容中都添加时间戳，用于标记本文消息发送的全局时间，同时在消息末尾增加 SAI 安全认证信息用于对消息来源的合法性进行预判。共享认证消息 SAI 的生成原理如图 6.3 所示。

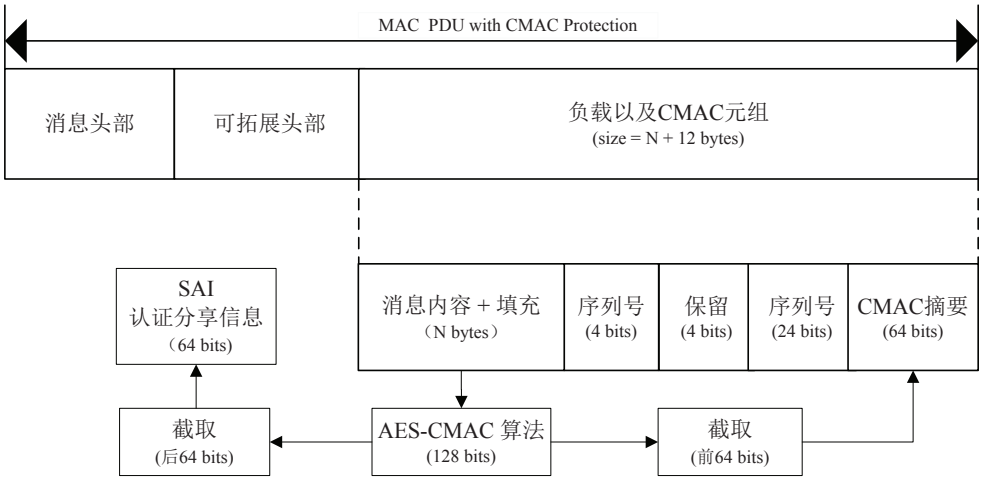

图 6.3: SAI 生成原理

在通过 CMAC 保护的消息中，由消息原文和 CMAC key 通过 AES-CMAC 算法生成 128 位数值，截取其中前 64 位作为 CMAC 消息验证码，而原本被丢弃的后 64 位则可被用作 SAI 共享认证信息。当目标站点接收到 CMAC 保护消息后，首先检测消息末尾的 SAI 是否与双方前一次通信消息生成的 SAI 一致，若结果一致则进行 CMAC 校验值的检测。SAI 的数值由前一次通信过程生成并预先保存，因此通过这种方式大大减少了由恶意 SS 引发的计算资源占用问题，能够有效阻止 DoS 攻击。最终 SS 与 BS 成功生成 TEK，实现消息 17 的加密传输。

6.2 密钥管理协议的建模和验证

通过 Maude 语言对改进后的密钥管理协议进行建模，本文中安全协议的形式化描述在原 PKMv3 协议的建模基础上进行模块的添加与修改，得以验证该协议所满足的安全特性。查看该协议是否会遭受到中间人攻击，重放攻击以及 DoS 攻击，从而使得协议机密性，完整性以及可用性遭到破坏。

6.2.1　安全协议建模过程

安全密钥管理协议中诚实主体的能力与 PKMv3 协议中相同，因此不再重新定义各站点的数据类型以及相关操作。协议双向认证阶段中 X.509 证书的请求与鉴别通过 CA 认证中心实现。在 Maude 中通过操作符`ca` 代表 CA 认证中心，并定义为`Ca` 数据类型。由 CA 认证中心发放的数字证书通过`ccert` 操作以站点名称为输入参数生成。为保护认证确认消息的完整性，重定义`checksum` 操作的输入参数，使得利用 AK 计算出消息原文的校验和。同时，定义`SAI` 数据类型代表共享认证信息，该数据类型由消息验证码 CMAC 生成，并将其设为`Cmac` 类型的子类。

改进后的协议同时采用随机值与时间戳保证 SS 与 BS 间会话的同步性。消息中的时间戳用于表明消息发送的具体时间，定义为`Tstamp` 数据类型，并通过自然数`Nat` 类型表示。定义`Curtime` 数据类型代表系统的当前时间，同样通过自然数`Nat` 表示。接收者可通过系统当前时间减去消息发送时间，计算出消息的真实传输时间，若计算时间比消息理论传输时间`tt` 的值大，则说明该消息可能由攻击者实施重放。因此，改进后的协议 SS 与 BS 间的连接状态可由四元组`{_,_,_,_}` 表示，输入参数分别为 AK 的生命周期，TEK 的生命周期，协议当前执行阶段和系统当前时间。

```
sorts Ca Curtime Tstamp SAI .
subsorts Nat < Curtime < Tstamp .
subsort Ca < Station . subsort SAI < Cmac .
op ca : -> Ca [ctor] .
op ccert : Station -> Cert .
op checksum : Contents Ak -> Checksum .
op sai : Cmac -> SAI .
op {_,_,_,_} : LTime LTime Process Curtime -> Systime .
```

在协议认证阶段，SS，BS 与 CA 间进行三方通信，包括 11 条认证消息。消息的发送与接收由重写规则定义。同时在安全组件交换以及传输密钥生成阶段，SS 与 BS 间分别传输了三条通信消息，消息中加入 SAI 认证用于抵御入侵者的 DoS 攻击，消息传输过程同样由重写规则描述。以下通过三条重写规则为例描述安全密钥管理协议各阶段站点间实施交互的过程。

定义`SendSSCertResponse` 规则描述 CA 认证中心接收到来自 SS 的数字

证书请求，并向其发送证书响应消息的过程。定义certssresponse 操作构造证书相应消息，包括 SS 的名称以及由 CA 公钥加密的 SS 证书，并将整条消息通过 CA 私钥加密实现数字签名。在认证密钥有效的情况下，每发送一次消息密钥的生命时长减少tt。连接参数中的变量CT 代表系统的当前时间，同理每发送一次消息，CT 增加tt 时长。

```
crl [SendSSCertResponse] :
  (node[A]: R0) (node[ca]: contnil)
  (from A to ca send MSG1 )
  {AKLT,TEKLT,pak,CT}
 =>
  (node[A]: R0) (node[ca]: ccert(A) )
  (from ca to A send sencrypt( certssresponse( A,encrypt(ccert(A),pubkey(ca)) )
      ,prikey(ca) ) )
  {sd(AKLT,tt),TEKLT,pak,(CT + tt)} if AKLT >= tt .
```

重写规则RecvAuthConfirm 用于描述 BS 接收消息池中认证确认消息的过程。消息接收前，消息池中存在认证确认消息MSG11 及其校验和。if 条件语句表述若从该消息中获取的时间戳与当前时间的绝对值为tt，且消息中通过getnon11 操作提取出的随机值与 BS 生成的随机值相等，接收到的消息校验和与利用 AK 计算出的校验和相等，BS 则从消息池中取出该条消息，并通过getaddr11 从该消息中操作提取出 SS 地址，从而计算出认证密钥 AK。

```
crl [RecvAuthConfirm] :
  (node[B]: PMK,PMKSN,nonce(A,B,R0),nonce(B,A,R1),CS1)
  (from A to B send (MSG11,CHECKSUM) ) {AKLT,TEKLT,pak,CT}
 =>
  (node[B]: PMK,PMKSN,nonce(B,A,R1),akcount(nonce(A,B,R0),nonce(B,A,R1)),
        getaddr11(MSG11),ak(PMK,B,getaddr11(MSG11),algo),
        checksum(MSG11,ak(PMK,B,getaddr11(MSG11),algo)),CS1)
  (msgnil) {aat,TEKLT,psa,CT}
  if (sd(CT,getct11(MSG11)) == tt) /\ getnon11(MSG11) == nonce(B,A,R1)
    /\ equalc( CHECKSUM,checksum(MSG11,ak(PMK,B,getaddr11(MSG11),algo)) ) .
```

定义重写规则SendThreeWayConfirm 描述 BS 向 SS 发送三次握手确认消息的过程。在 BS 发送三次握手确认消息前，它的消息集中预存了与 SS 前一次消息传输计算的共享认证信息SAI1。与 PKMv3 协议中三次握手过程相比，由 CMAC 保护的消息内容中添加时间戳CT，由系统当前时间表示。同时，在消息末尾添加共享认证信息SAI1，便于接收者对消息来源实行预判。消息成功发送后，AK 生

命时长相应减少tt, 而系统当前时间增加tt 时长。

```
crl [SendThreeWayConfirm] :
  (node[B]: nonce(B,A,R1),CMACKEY,PMKSN,nonce(A,B,R0),SNP,SECU,SAI1,CS1)
  (msgnil) {AKLT,TEKLT,psa,CT}
 =>
  (node[B]: nonce(B,A,R1),CMACKEY,PMKSN,nonce(A,B,R0),SNP,SECU,SAI1,CS1,
      said(nonce(B,A,R1),nonce(A,B,R0),SECU) )
  (from B to A send (threewayconfirm(nonce(A,B,R0),PMKSN,said(nonce(B,A,R1),
     nonce(A,B,R0),SECU),SNP,CT),
    cmac(threewayconfirm(nonce(A,B,R0),PMKSN,said(nonce(B,A,R1),nonce(A,B,R0),
       SECU),SNP,CT),CMACKEY),SAI1))
  {sd(AKLT,tt),TEKLT,psa,(CT + tt)}
  if AKLT >= tt /\ not( said(nonce(B,A,R1),nonce(A,B,R0),SECU) inc CS1) .
```

6.2.2　安全协议验证过程

针对 PKMv3 协议中遭受入侵者实施窃取，重放以及窃取等行为引发的协议安全漏洞，通过search 指令搜索安全密钥管理协议的状态空间，以检测改进后的协议是否能够抵抗 DoS，中间人以及重放等攻击，从而保证协议的完整性，机密性以及可用性等安全性质。

PKMv3 协议存在消息完整性缺陷，即 SS 与 BS 间传输的认证确认消息可能会遭到篡改。通过以下search 指令验证 BS 接收到的认证确认消息中是否存在错误 SS 地址信息。执行结果为无此路径，表明 BS 不会接收到错误的地址信息，通过 AK 作为操作算子计算出的消息校验和能够有效的保护认证确认消息的完整性。

```
Maude> search init0 =>* S:States node[bs("b")]: C:Contents,A:Ssaddr,K:Ak such
    that A:Ssaddr =/= ssaddr(ss("a")) = true .

search in INTRUDERS : init0 =>* S:States node[bs("b")]: A:Ssaddr,C:Contents,K:Ak
     such that A:Ssaddr =/= ssaddr(ss("a")) = true .

No solution.
states: 905 rewrites: 12427 in 96ms cpu (98ms real) (129447 rewrites/second)
```

由 PKMv3 协议验证过程可知，原协议虽然能够保证协议密钥的机密性，但入侵者仍有可能窃取到明文证书信息。通过search 指令检测任意系统状态下入侵者消息集中存在 SS 数字证书或 BS 数字证书的情况。执行结果依然返回无此路径，表明通过 CA 认证中心作为第三方实施基站与手机站间的认证过程，能够有效保护双方数字证书的机密性，使之无法被入侵者窃取。

```
Maude> search init0 =>* S:States node[intru("c")]: C:Contents such that (ccert(
    ss("a")) inc C:Contents) or (ccert(bs("b")) inc C:Contents) .

search in INTRUDERS : init0 =>* S:States node[intru("c")]: C:Contents such that
     ccert(ss("a")) inc C:Contents or ccert(bs("b")) inc C:Contents = true .

No solution.
states: 905 rewrites: 17849 in 92ms cpu (95ms real) (194010 rewrites/second)
```

入侵者对 PKMv3 协议中 SS 与 BS 间的消息传输实施中间人攻击，不断获取并重放站点间的传输消息，此类行为虽未损害协议执行的完整性与认证性，多次转发过程大大损害了协议密钥的生命时长。通过以下search 指令检测入侵者获取 SAID 等相关信息所执行的重写路径。执行结果显示一共存在四条可执行路径，并通过show path labels [n] 指令打印各路径所执行的重写规则，由于其中并不包括代表入侵者重放行为的重写规则，表明通过在消息中插入时间戳能够有效识别入侵者的重放消息并将其丢弃，保障了密钥的可用性。

```
Maude> search init0 =>! (S:States) (node[intru("c")]: C:Contents, said(nonce(bs
    ("b"),ss("a"),next(seed1)),nonce(ss("a"),bs("b"),next(seed)),secucapa(ss("a"))
    )) .

search in INTRUDERS : init0 =>! S:States node[intru("c")]: C:Contents,said(nonce
    (bs("b"),ss("a"),next(seed1)),nonce(ss("a"),bs("b"),next(seed)),secucapa(ss("a
    "))) .

Solution 1 (state 801)
states: 807 rewrites: 10749 in 100ms cpu (102ms real) (107490 rewrites/second)
...

Solution 2 (state 806)
states: 811 rewrites: 10783 in 100ms cpu (103ms real) (107830 rewrites/second)
...

Solution 3 (state 809)
states: 813 rewrites: 10801 in 100ms cpu (103ms real) (108010 rewrites/second)
...

Solution 4 (state 810)
states: 813 rewrites: 10803 in 100ms cpu (104ms real) (108030 rewrites/second)
S:States --> msgnil msgs {60,60,pak,0}
(node[ca]: contnil) (node[ss("a")]: seed) (node[bs("b")]: seed1)
C:Contents --> seed0, ssaddr(ss("a")), nexte(nexte(eks)),
flag(0),snp(64,1000), secucapa(ss("a")),
pmksn(pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1
    ),algo)),
encrypt(ccert(ss("a")),pubkey(ca)),encrypt(ccert(bs("b")),
nonce(ss("a"), bs("b"),seed), nonce(bs("b"),ss("a"),seed1),pubkey(ca)),
nonce(ss("a"),bs("b"),next(seed)), nonce(bs("b"),ss("a"),next(seed1)),
```

```
countertek(nonce(ss("a"),bs("b"),next(seed)),nonce(bs("b"),ss("a"),next(seed1)),
    nexte(nexte(eks))),
akid(ak(pmk(ss("a"), bs("b"),nonce(ss("a"),bs("b"),seed),nonce(bs("b"),ss("a"),
    seed1),algo),bs("b"),ssaddr(ss("a")), algo),pmksn(pmk(ss("a"),bs("b"),nonce(ss
    ("a"),bs("b"),seed),nonce(bs("b"),ss("a"),seed1),algo)),ss("a"),bs("b"),algo)

No more solutions.
states: 905 rewrites: 12419 in 108ms cpu (112ms real) (114990 rewrites/second)
```

6.3　现有改进方案对比与分析

目前针对 WiMAX 中密钥管理协议出现的各类安全漏洞，本文以及国内外研究者均提出了不少改进方案。表 6.1 总结并分析了现有的密钥管理协议改进方案，根据它们所解决的问题，进行改造的协议阶段，优缺点等方面一一进行阐述。在该表格中，前四条为本文所采用的改进方法，涉及协议执行的各个阶段，有效抵御 Replay 和 DoS 攻击，保证消息的完整性与机密性。在实施这些方案的同时，需要原协议标准的相应改变。如加入时间戳必须实施并支持消息的同步，通过 AK 计算校验和会增加计算复杂度，通过 CA 作为第三方认证相应的增加了认证时间。采用 SAI 共享认证信息进行消息预判的方式，同样会增加信息存储空间。

除此之外，针对在认证过程中 BS 可能遭遇恶意请求站点实施 DoS 攻击的行为，可实施基于视觉密码的认证，双方通过共享的二进制图像以及登记的可信第三方实施这一过程。该改进方案的缺陷为视觉密码认证方式需要认证标准的完全改变。在密钥管理协议的认证过程中，协议标准通过 RSA 算法实施公钥加密，该加密算法可由 ECC 公钥加密算法替代。ECC 算法相较 RSA 拥有更快的计算速度以及更高的安全性，这一改进方案同样需要协议标准的相应改变。由于 PKMv3 协议中认证密钥以及传输密钥的序列号均很小，可能会遭遇入侵者攻击在会话中插入重利用的密钥，从而获取传输消息，通过增加密钥空间可改善这一缺陷，但需要硬件支持以及协议标准的改变。SINEP 方案提出手机端进入 WiMAX 网络初始化过程的方案，该方案基于 DH 密钥交换原理，可有效阻止中间人攻击，但仍存在提出太多假设以及提高计算成本的缺点。

表 6.1: 协议改进方案对比与分析

改进方案	解决问题	应用阶段	优势	劣势
随机值 + 时间戳	抵御 Replay 和 DoS 攻击	整个协议阶段	阻止穿插攻击和 Dos 攻击	需要实施消息同步
AK 计算校验和	保证完整性	认证确认消息	能够检测出消息误传与篡改	增加计算复杂度
CA 认证中心	保证机密性	认证阶段	站点证书不会遭到泄露	增加认证时间
SAI 共享密钥信息	抵御 Dos 攻击	CMAC 保护阶段	通过信息预判减少计算负担	需要额外空间存储 SAI 信息
视觉密码认证	抵御 DoS 攻击	认证阶段	阻止来自恶意 SS 的请求	需要认证标准的完全改变
ECC 加密算法	加密算法计算效率	协议加密算法	减少密钥空间和计算量	需要协议标准的改变
增加密钥空间	改进密钥空间缺陷	AK 与 TEK 的序列号	阻止循环密钥空间攻击	需要硬件升级，改变协议标准
SINEP 方案	初次进入网络认证方案	认证阶段	能够阻止中间人攻击	提出大量假设，增加计算负担

6.4 本章小结

本章针对前一章节中总结的 PKMv3 各阶段出现的安全缺陷，依次提出改进方案，并构造出一种安全的密钥管理协议。改进后的密钥管理协议通过 CA 认证中心作为第三方实施站点间的认证，实现数字证书的机密性以及认证的可靠性。并通过随机值与时间戳混合保护的方式保障消息的同步传输，有效预防网络中消息重放的攻击行为。与此同时，通过 AK 作为操作算子计算消息校验和预防入侵者篡改消息的行为，并提出基于安全认证消息 SAI 的 CMAC 消息保护方式，其通过预判断的方式减少系统计算负担，预防 DoS 攻击。通过 Maude 进行安全密钥管理协议的建模与分析，验证得出改进后的协议能够弥补 PKMv3 中各安全漏洞。

第七章　总结和展望

7.1　论文总结

在准 4G 标准 WiMAX 技术的应用日益广泛的背景下，对 IEEE802.16m 标准安全子层中隐私保护功能的可靠性要求逐渐增高。基于目前针对 IEEE802.16m 标准中密钥管理 PKMv3 协议安全性能的研究仍然不够完善的现状，本文详细分析与阐述了协议的理论基础以及协议执行各阶段的具体步骤，并通过基于重写逻辑的可执行形式化规范语言 Maude 实现对 PKMv3 协议执行过程的形式化建模。同时，通过分析 PKMv3 协议网络环境中入侵者的能力，通过重写规则模拟攻击者的行为。在定义完成的 PKMv3 协议模型的基础上，利用 Maude 自带的验证方法以及模型检测工具，实现对协议时间特性和安全特性的验证。最终，本文针对协议各阶段存在的安全漏洞提出相应的解决方案，并重新对改进后的协议进行建模验证，证明完善后协议安全性能的可靠性。

本文的主要工作分为以下三点：

首先，通过 Maude 语言描述 PKMv3 网络结构，并通过重写规则实现对协议各执行阶段，包括双向认证阶段，安全组件协商阶段，传输密钥生成阶段，加密消息传输阶段中基站与手机站间消息传递的建模。同时，在协议模型中引入时间机制，模拟密钥的生命周期，实现新旧密钥的不断迭代和通信过程的持续执行。此外，在 PKMv3 网络环境中引入攻击者模型，模拟协议各执行阶段入侵者对基站与手机站间的通信消息进行截取，重放或者伪造的攻击行为。

其次，利用 Maude 提供的 LTL 模型检测工具实现对 PKMv3 可执行规范进行检测，验证协议的时间特性，如执行连续性，AK 以及 TEK 生命周期，密钥新鲜性

等性质。通过 Maude 自带的 search 指令，穷尽协议系统的可达状态，验证协议的安全特性，如机密性，认证性，机密性以及可用性等性质。验证结果表明，本文建立的协议模型能够满足协议标准中定义的时间特性，但存在无法保证消息传输机密性以及系统资源可用性等方面的安全漏洞。

最后，根据本文以及其他学者提出的 PKMv3 协议缺陷，提出了改进后的密钥管理协议，详细定义协议各阶段消息传输的具体内容。引入 CA 认证中心实现数字证书的分发与验证，添加时间戳从而抵御重放攻击，并通过计算共享认证信息以抵御 DoS 攻击。本文将改进后的协议在原协议模型的基础上进行修改，从而验证得出新的密钥管理协议能够弥补各项安全漏洞。

7.2 工作展望

本文利用 Maude 语言实现对 PKMv3 协议的形式化建模与验证。针对 PKMv3 模型中存在密钥生命周期迭代的时间机制，本文通过自定义的时间类型以及相关操作，实现协议执行过程中认证密钥与传输密钥生命周期的模拟。Real-Time Maude[59] 在 Maude 语言的基础上拓展了时间机制，通过预定义数据类型以及相关操作，能够支持对实时系统的建模。因此，对 PKMv3 协议的研究可通过 Real-Time Maude 语言进行建模，Real-Time Maude 定义了全局系统时间，并提供更加的实施重写，通过时间约束的状态搜索指令以及模型检测工具。在另一方面，由于 LTL 模型检测限制，本文所描述的协议状态空间是有限的，可以通过 Maude LTL LBMC[60] 工具实施对真实情况下无限状态空间协议的检测。

Maude 语言同样支持面向对象的建模方式。在面向对象的系统模块中，参与协议通信过程的实体被描述为对象，可以通过定义相关类的属性实现对象属性的继承。系统中各对象间通过预定义的消息类型进行交互，通信过程由面向对象的重写规则实现状态转化。因此，在后续的研究中，可以使用面向对象的建模方式，实现对 PKMv3 协议中不同基站间快速切换机制的建模，以及实现对分布式数据库协议，组播传输协议等更多安全协议的形式化规范描述。

附录 A　重写规则与变量表

描述 PKMV3 通信过程的重写规则

```
crl [RecvAuthRequestMsgReal] :
   (node[B]: R1,CERT_B )
   (from A to B send sencrypt( MSG1,PRI1) ) {AKLT,TEKLT,pak} =>
   (node[B]: R1,CERT_B,getcert1(MSG1),getnon1(MSG1) )
   (msgnil) {AKLT,TEKLT,pak}
   if getpub( getcert1( sdecrypt(sencrypt(MSG1,PRI1),pubkey(A))) ) == pubkey(A) .

crl [SendAuthResponseMsgReal] :
   (node[B]: R1,CERT_B,CERT_A,nonce(A,B,R0) )
   (msgnil) {AKLT,TEKLT,pak} =>
   (node[B]: next(R1),CERT_B,CERT_A,nonce(A,B,R0),nonce(B,A,R1),
      pmk(A,B,nonce(A,B,R0),nonce(B,A,R1),algo ),pmksn( pmk(A,B,nonce(A,B,R0),
      nonce(B,A,R1),algo)),akcount(nonce(A,B,R0),nonce(B,A,R1)))
   (from B to A send sencrypt( authresponse(CERT_B,nonce(A,B,R0),nonce(B,A,R1),
       encrypt( pmk(A,B,nonce(A,B,R0),nonce(B,A,R1),algo ),pubkey(A) ),
       pmksn(pmk(A,B,nonce(A,B,R0),nonce(B,A,R1),algo))), prikey(B) ) )
   {sd(AKLT,tt),TEKLT,pak} if AKLT >= tt .

crl [SendAuthConfirmMsgReal] :
   (node[A]: next(R0),nonce(A,B,R0),nonce(B,A,R1),PMK,PMKSN,CS1)
   (msgnil) {AKLT,TEKLT,pak} =>
   (node[A]: next(R0),nonce(A,B,R0),nonce(B,A,R1),PMK,PMKSN,CS1,ssaddr(A),
       akcount(nonce(A,B,R0),nonce(B,A,R1)),ak(PMK,B,ssaddr(A),algo) )
   (from A to B send (authconfirm( nonce(B,A,R1),ssaddr(A) ),
       checksum( authconfirm( nonce(B,A,R1),ssaddr(A),icv) ) )
   {sd(AKLT,tt),TEKLT,pak} if AKLT >= tt /\ not(ssaddr(A) inc CS1) .

crl [RecvAuthConfirmMsgReal] :
   (node[B]: nonce(A,B,R0),nonce(B,A,R1),PMK,AKCOUNT,CS2)
   (from A to B send (MSG3,CHECKSUM) ) {AKLT,TEKLT,pak} =>
   (node[B]: nonce(A,B,R0),nonce(B,A,R1),PMK,AKCOUNT,CS2,
      getssaddr(MSG3), ak(PMK,B,getssaddr(MSG3),algo) )
   (msgnil) {aat,TEKLT,pak}
   if equals( getnon3(MSG3),nonce(B,A,R1) ) == true /\
     equalc( CHECKSUM,checksum(MSG3,icv) ) .

crl [RecvThreeWayRequestMsgReal] :
   (node[A]: R0,PMKSN,AK,AKCOUNT,CS2)
   (from B to A send (MSG4,CMAC1)) {AKLT,TEKLT,psa} =>
   (node[A]: R0,PMKSN,AK,AKCOUNT,getnon4(MSG4),getakid4(MSG4),
     cmactek(AK,AKCOUNT,algo),cmackey(cmactek(AK,AKCOUNT,algo),algo) )
   (msgnil) {AKLT,TEKLT,psa}
```

```
43    if getpmksn4(MSG4) == PMKSN /\ equalm(cmac(MSG4,cmackey(cmactek(AK,AKCOUNT,
          algo),algo)),CMAC1) .
44
45  crl [SendThreeWayResponseReal] :
46    (node[A]: next(R0),PMKSN,nonce(B,A,R1),AKID,CMACKEY,CS1)
47    (msgnil) {AKLT,TEKLT,psa} =>
48    (node[A]: next(next(R0)),PMKSN,nonce(B,A,R1),AKID,CMACKEY,CS1,
49       nonce(A,B,next(R0)),snp(64,1000),secucapa(A) )
50    (from A to B send (threewayresponse(nonce(B,A,R1),nonce(A,B,next(R0)),PMKSN,
51       AKID,snp(64,1000),secucapa(A)),cmac(threewayresponse(nonce(B,A,R1),
52       nonce(A,B,next(R0)),PMKSN,AKID,snp(64,1000),secucapa(A)),CMACKEY) ) )
53    {sd(AKLT,tt),TEKLT,psa}
54    if AKLT >= tt /\ not(snp(64,1000) inc CS1) .
55
56  crl [SendThreeWayConfirmReal] :
57    (node[B]: nonce(B,A,R1),PMKSN,CMACKEY,nonce(A,B,R0),SNP,SECU,CS1)
58    (msgnil) {AKLT,TEKLT,psa} =>
59    (node[B]: nonce(B,A,R1),PMKSN,CMACKEY,nonce(A,B,R0),SNP,SECU,CS1,
60      said(nonce(B,A,R1),nonce(A,B,R0),SECU),eks)
61    (from B to A send (threewayconfirm(nonce(A,B,R0),PMKSN,SNP,said(nonce(B,A,R1)
          ,nonce(A,B,R0),SECU)),cmac(threewayconfirm(nonce(A,B,R0),
62        PMKSN,SNP,said(nonce(B,A,R1),nonce(A,B,R0),SECU)),CMACKEY)) )
63    {sd(AKLT,tt),TEKLT,psa} if AKLT >= tt .
64
65  crl [RecvThreeWayConfirmReal] :
66    (node[A]: PMKSN,nonce(A,B,R0),CMACKEY,SNP,SECU,CS2)
67    (from B to A send (MSG6,CMAC1)) {AKLT,TEKLT,psa} =>
68    (node[A]: PMKSN,nonce(A,B,R0),CMACKEY,SNP,SECU,CS2,getsaid6(MSG6))
69    (msgnil) {AKLT,TEKLT,ptek}
70    if getpmksn6(MSG6) == PMKSN /\ equalm(cmac(MSG6,CMACKEY),CMAC1) /\
71       equals(getnon6(MSG6),nonce(A,B,R0)) .
72
73  crl [RecvTEKRequestMsgReal] :
74    (node[B]: PMKSN,CMACKEY,SAID,EKS,CS2 )
75    (from A to B send (MSG7,CMAC1) ) {AKLT,TEKLT,ptek} =>
76    (node[B]: PMKSN,CMACKEY,SAID,EKS,CS2,getflag(MSG7) )
77    (msgnil) {AKLT,TEKLT,ptek}
78    if getsaid7(MSG7) == SAID /\ getpmksn7(MSG7) == PMKSN
79    /\ equalm(cmac(MSG7,CMACKEY),CMAC1) .
80
81  crl [SendTEK0ResponseMsgReal] :
82    (node[B]: nonce(B,A,R1),nonce(A,B,R0),PMKSN,CMACTEK,CMACKEY,SAID,flag(0),EKS,
          CS2 )
83    (msgnil) {AKLT,TEKLT,ptek} =>
84    (node[B]: nonce(B,A,R1),nonce(A,B,R0),PMKSN,CMACTEK,CMACKEY,SAID,flag(0),
85       nexte(EKS),CS2,countertek(nonce(A,B,R0),nonce(B,A,R1)),
86     tek1(CMACTEK,SAID,countertek(nonce(A,B,R0),nonce(B,A,R1)),algo),
87     tek2(CMACTEK,SAID,nextc(countertek(nonce(A,B,R0),nonce(B,A,R1))),algo) )
88    (from B to A send (tekresponse(SAID,PMKSN,countertek(nonce(A,B,R0),
89       nonce(B,A,R1)),EKS),cmac(tekresponse(SAID,PMKSN,countertek(nonce(A,B,R0),
90       nonce(B,A,R1)),EKS),CMACKEY) ) )
91    {sd(AKLT,tt),sd(TEKLT,tt),ptek}
92    if not(countertek(nonce(A,B,R0),nonce(B,A,R1)) inc CS2)
93    /\ AKLT >= tt /\ TEKLT >= tt .
94
```

```
crl [SendTEK1ResponseMsgReal] :
  (node[B]: nonce(B,A,R1),nonce(A,B,R0),PMKSN,CMACTEK,CMACKEY,SAID,flag(1),TEK,
     EKS,CS2,C_TEK )
  (msgnil) {AKLT,TEKLT,ptek} =>
  (node[B]: nonce(B,A,R1),nonce(A,B,R0),PMKSN,CMACTEK,CMACKEY,SAID,flag(1),TEK,
    nexte(EKS),CS2,C_TEK,tek2(CMACTEK,SAID,C_TEK,algo) )
  (from B to A send (tekresponse(SAID,PMKSN,countertek(nonce(A,B,R0),
    nonce(B,A,R1)),EKS),cmac(tekresponse(SAID,PMKSN,countertek(nonce(A,B,R0),
    nonce(B,A,R1)),EKS),CMACKEY) ) )
  {sd(AKLT,tt),sd(TEKLT,tt),ptek}
   if AKLT >= tt /\ TEKLT >= tt /\ not(tek2(CMACTEK,SAID,C_TEK,algo) inc CS2) .

crl [RecvTEK1ResponseMsgReal] :
  (node[A]: PMKSN,CMACTEK,CMACKEY,SAID,FLAG,CS1)
  (from B to A send (MSG8,CMAC2)) {AKLT,TEKLT,ptek} =>
  (node[A]: PMKSN,CMACKEY,SAID,FLAG,CS1,getcontek(MSG8),geteks(MSG8),
    tek2(CMACTEK,SAID,getcontek(MSG8),algo) )
  (msgnil) {AKLT,tat,ptek}
  if getsaid8(MSG8) == SAID /\ getpmksn8(MSG8) == PMKSN /\
   equalm(cmac(MSG8,CMACKEY),CMAC2) /\ FLAG == flag(1) .
```

描述入侵者行为的重写规则

```
crl [SendAuthRequestMsgFake] :
  (node[C]: R2,CERT_C,CERT_A,nonce(A,B,R0) )
  (node[B]: R1,CERT_B )
  (msgnil) {AKLT,TEKLT,PS} =>
  (node[C]: R2,CERT_C,CERT_A,nonce(A,B,R0) )
  (node[B]: R1,cert(B,macaddr(B),pubkey(B) ) )
  (from A to B send sencrypt(authrequest(CERT_A,nonce(A,B,R0),capa(A)),
      prikey(A)))
  {sd(AKLT,tt),TEKLT,PS} if AKLT > tt .

crl [SendAuthRequestMsgChange] :
  (node[C]: R2,CERT_C,CERT_A,nonce(A,B,R0) )
  (node[B]: R1,CERT_B)
  (msgnil) {AKLT,TEKLT,PS} =>
  (node[C]: R2,CERT_C,CERT_A,nonce(A,B,R0) )
  (node[B]: R1,CERT_B )
  (from A to B send sencrypt(authrequest(CERT_C,nonce(A,B,R0),capa(C)),
      prikey(C)))
  {sd(AKLT,tt),TEKLT,PS} if AKLT > tt .

crl [RecvAuthResponseMsgFake] :
  (node[C]: R2,CERT_C,CERT_A,nonce(A,B,R0) )
  (from B to A send sencrypt( AUTHRESPONSE,PRI2 ) ) =>
  (node[C]: R2,CERT_C,CERT_A,nonce(A,B,R0),getcert2(AUTHRESPONSE),
    getnon22(AUTHRESPONSE),decrypt(getcipher(AUTHRESPONSE),prikey(C) ),
    getpmksn2(AUTHRESPONSE) ) (msgnil)
  if equals( getnon21(AUTHRESPONSE),nonce(A,B,R0)) == true /\ getpub(
     getcert2(sdecrypt(sencrypt(AUTHRESPONSE,PRI2),pubkey(B)))) == pubkey(B) .

crl [SendAuthResponseMsgChange] :
```

```
   (node[A]: next(R0),CERT_A,nonce(A,B,R0))
   (node[C]: R2,CERT_C,CERT_A,CERT_B,nonce(A,B,R0),nonce(B,A,R1),NC1,PMKSN)
   (msgnil) {AKLT,TEKLT,PS} =>
   (node[A]: next(R0),CERT_A,nonce(A,B,R0))
   (node[C]: R2,CERT_C,CERT_A,CERT_B,nonce(A,B,R0),nonce(B,A,R1),NC1,PMKSN)
   (from B to A send sencrypt(authresponse(CERT_C,nonce(A,B,R0),nonce(B,A,R1),
       encrypt(pmk(A,B,nonce(A,B,R0),nonce(B,A,R1),algo),pubkey(A)),PMKSN),
          prikey(C)))
   {sd(AKLT,tt),TEKLT,PS} if AKLT > tt .

crl [RecvAuthConfirmMsgFake] :
   (node[C]: R2,CERT_C,CERT_A,CERT_B,nonce(A,B,R0),nonce(B,A,R1),NC1,PMKSN)
   (from A to B send (AUTHCONFIRM,CHECKSUM) ) =>
   (node[C]: R2,CERT_C,CERT_A,CERT_B,nonce(A,B,R0),nonce(B,A,R1),
      NC1,PMKSN,getssaddr(AUTHCONFIRM) ) (msgnil)
   if equals(getnon3(AUTHCONFIRM),nonce(B,A,R1)) == true /\
     equalc(CHECKSUM,checksum(AUTHCONFIRM,icv)) .

crl [SendAuthConfirmMsgFake] :
  (node[B]: next(R1),CERT_B,CERT_A,nonce(A,B,R0),nonce(B,A,R1),
      PMK,PMKSN,AKCOUNT)
  (node[C]: R2,CERT_C,CERT_A,CERT_B,nonce(A,B,R0),nonce(B,A,R1),
      NC1,PMKSN,ssaddr(A))
  (msgnil) {AKLT,TEKLT,PS} =>
  (node[B]: next(R1),CERT_B,CERT_A,nonce(A,B,R0),nonce(B,A,R1),
      PMK,PMKSN,AKCOUNT)
  (node[C]: R2,CERT_C,CERT_A,CERT_B,nonce(A,B,R0),nonce(B,A,R1),
      NC1,PMKSN,ssaddr(A) )
  (from A to B send (authconfirm(nonce(B,A,R1),ssaddr(A)),
     checksum( authconfirm(nonce(B,A,R1),ssaddr(A)),icv)))
  {sd(AKLT,tt),TEKLT,PS} if AKLT > tt .

crl [ReplayThreeWayRequestMsg] :
   (node[C]: PMKSN,ssaddr(A),CS2)
   (from B to A send (MSG4,CMAC1)) {AKLT,TEKLT,psa} =>
   (node[C]: PMKSN,ssaddr(A),CS2,getnon4(MSG4),getakid4(MSG4) )
   (from B to A send (MSG4,CMAC1)) {AKLT,TEKLT,psa}
   if getpmksn4(MSG4) == PMKSN .

crl [ReplayThreeWayResponse] :
   ( node[C]: nonce(B,A,R1),AKID,CS1)
   ( from A to B send (MSG5,CMAC2) ) {AKLT,TEKLT,psa} =>
   ( node[C]: nonce(B,A,R1),AKID,CS1,
       getnon52(MSG5),getsnp5(MSG5),getsc(MSG5))
   ( from A to B send (MSG5,CMAC2) ) {AKLT,TEKLT,psa}
   if getakid5(MSG5) == AKID /\ equals(getnon51(MSG5),nonce(B,A,R1)) .

crl [ReplayTEKRequestMsgReal] :
   (node[C]: PMKSN,SAID,CS2 )
   (from A to B send (MSG7,CMAC1) ) {AKLT,TEKLT,ptek} =>
   (node[C]: PMKSN,SAID,CS2,getflag(MSG7) )
   (from A to B send (MSG7,CMAC1) ) {AKLT,TEKLT,ptek}
   if getsaid7(MSG7) == SAID /\ getpmksn7(MSG7) == PMKSN .
```

表 A.1: 重写规则中出现的变量类型以及含义

变量	类型	描述
A, B, C	Ss, Bs, Intruder	手机站, 基站, 侵入者
R0, R1, R2,	Rand	各主体的初始随机数
CERT_X(X=A, B, C)	Cert	主体 X 的数字证书
PRI1, PRI2	Prikey	各主体的私钥
PMK, PMKSN	Pmk, Pmksn	主密钥，主密钥的序列号
AK, AKID	Ak, AkId	认证密钥，认证密钥的标识符
AKCOUNT	Akcount	维护认证密钥的计数值
CMACTEK, CAMCKEY	Cmackey, CmacTek	产生传输密钥，消息验证码的密钥
SECU, SNP	SecuCapa, SNP	安全能力，安全协商参数
SAID	SaId	安全组件标识符
EKS, FLAG	EKS, TRFlag	加密密钥序列号，密钥刷新标识
TEK, C_TEK	Tek, CounterTek	传输密钥，维护 TEK 的计数值
CIPHER, NC1	Cipher, Ncipher	加密，解密后的内容
CHECKSUM	Checksum	消息校验和
CMAC1, CMAC2	Cmac	消息验证码
CS1, CS2	Contents	消息内容
AKLT, TEKLT	LTime	AK 以及 TEK 的生命周期
PS	Process	协议当前执行阶段
MSG1	AuthRequest	认证请求消息
MSG2	AuthResponse	认证响应消息
MSG3	AuthConfirm	认证确认消息
MSG4	ThreeWayRequest	三次握手请求消息
MSG5	ThreeWayResonse	三次握手响应消息
MSG6	ThreeWayConfirm	三次握手确认消息
MSG7	TekRequest	密钥请求消息
MSG8	TekResponse	密钥响应消息
MSG9	TekInvalid	请求失效消息

参考文献

[1] IEEE B E. IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Broadband Wireless Access Systems Amendment 3: Advanced Air Interface[J], 2011 : 1 – 1112.

[2] ANDREWS J G, GHOSH A, MUHAMED R. Fundamentals of WiMAX: Understanding Broadband Wireless Networking[J]. Prentice Hall Communications Engineering & Emerging Technologies, 2007.

[3] 李凤海. 无线城域网安全子层分析与研究 [D]. [S.l.] : 国防科学技术大学, 2006.

[4] 鲁来凤. 安全协议形式化分析理论与应用研究 [D]. [S.l.] : 西安电子科技大学, 2012.

[5] CLAVEL M, FRANCISCO N, EKER S, et al. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic[C] // . 2007.

[6] 吴昌. 网络安全协议的高效分析系统 [D]. [S.l.] : 南昌大学, 2008.

[7] XU S, HUANG C T. Attacks on PKM Protocols of IEEE 802.16 and Its Later Versions[C] // International Symposium on Wireless Communication Systems. 2006 : 185 – 189.

[8] KAHYA N, GHOUALMI N, LAFOURCADE P. Formal analysis of PKM us-

ing scyther tool[C] // International Conference on Information Technology and E-Services. 2012 : 1 – 6.

[9] KAHYA N, GHOUALMI N, LAFOURCADE P. Secure Key Management Protocol in WIMAX[J]. International Journal of Network Security & Its Applications, 2012, 4(6) : 119 – 132.

[10] YOU Z, XIE X, ZHENG W. Verification and research of a Wimax authentication protocol based on SSM[C] // Education Technology and Computer (ICETC), 2010 2nd International Conference on. 2010 : V5 – 238.

[11] RAI A K, MISHRA S, TRIPATHI P N. An Improved Secure Authentication Protocol for WiMAX with Formal Verification[C] // Advances in Computing and Communications - First International Conference, ACC 2011, Kochi, India, July 22-24, 2011. Proceedings. 2011 : 407 – 416.

[12] XU S X S, HUANG C T, MATTHEWS M M. Modeling and analysis of IEEE 802.16 PKM Protocols using CasperFDR[J], 2008 : 653 – 657.

[13] RAJU K V K. Formal Verification of IEEE802.16m PKMv3 Protocol Using CasperFDR[C] // Information and Communication Technologies - International Conference, ICT 2010, Kochi, Kerala, India, September 7-9, 2010. Proceedings. 2010 : 590 – 595.

[14] ZHU X, XU Y, GUO J, et al. Formal Verification of PKMv3 Protocol Using DT-Spin[C] // International Symposium on Theoretical Aspects of Software Engineering. 2015 : 71 – 78.

[15] SIKKENS B. Security issues and proposed solutions concerning authentication and authorization for WiMAX (IEEE 802.16 e)[C] // . 2008.

[16] HAN T, ZHANG N, LIU K, et al. Analysis of mobile WiMAX security: Vulnerabilities and solutions[C] // Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on. 2008 : 828 – 833.

[17] BOGDANOSKI M, LATKOSKI P, RISTESKI A, et al. IEEE 802.16 Security Issues: A Survey[C] // Telecommunications Forum Telfor. 2008.

[18] TSHERING F, SARDANA A. A Review of Privacy and Key Management Protocol in IEEE 802.16e[J]. International Journal of Computer Applications, 2011, 20(20) : 25 – 31.

[19] TIAN H, PANG L, WANG Y. Key Management Protocol of the IEEE 802.16e[J]. Wuhan University Journal of Natural Sciences, 2007, 12(1) : 59 – 62.

[20] 杨玖宏, 王玉柱, 何定养. IEEE802.16m 中 PKMv3 协议的安全分析及改进 [J]. 中国储运, 2012(8) : 126 – 129.

[21] HASHMI R M, SIDDIQUI A M, JABEEN M, et al. Improved Secure Network Authentication Protocol (ISNAP) for IEEE 802.16[C] // International Conference on Information and Communication Technologies. 2009 : 101 – 105.

[22] ALTAF A, JAVED M Y, AHMED A. Security Enhancements for Privacy and Key Management Protocol in IEEE 802.16e-2005[C] // Acis International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/distributed Computing. 2008 : 335 – 339.

[23] ALTAF A, SIRHINDI R, AHMED A. A Novel Approach against DoS Attacks in WiMAX Authentication Using Visual Cryptography[C] // International Conference on Emerging Security Information, Systems and Technologies, 2008. Securware. 2008 : 238 – 242.

[24] KIM Y, LIM H K, BAHK S. Shared Authentication Information for Preventing DDoS attacks in Mobile WiMAX Networks[C] // Consumer Communications and NETWORKING Conference, 2008. Ccnc. 2008 : 765 – 769.

[25] FU A, ZHANG Y, ZHU Z, et al. A Fast Handover Authentication Mechanism Based on Ticket for IEEE 802.16m[J]. Communications Letters IEEE, 2010, 14(12) : 1134 – 1136.

[26] SHRESTHA S L, SONG N O, CHONG S. Seamless realtime traffic handover policy for IEEE 802.16m mobile WiMAX[C] // Information Sciences and Systems, 2009. Ciss 2009. Conference on. 2009 : 252 – 257.

[27] PITA I, RIESCO A. Specifying and Analyzing the Kademlia Protocol in Maude[M]. [S.l.] : Springer International Publishing, 2015.

[28] RODRíGUEZ A R, LóPEZ J A V. The EIGRP Protocol in Maude[J]. Informe Técnico, 2007.

[29] OLVECZKY P C, PRABHAKAR P, LIU X. Formal modeling and analysis of real-time resource-sharing protocols in Real-Time Maude[J]. Journal of Separation Science, 2008, 33(4-5) : 1 – 8.

[30] LVECZKY P C, KEATON M, MESEGUER J, et al. Specification and Analysis of the AER/NCA Active Network Protocol Suite in Real-Time Maude[C] // International Conference on Fundamental Approaches To Software Engineering. 2001 : 333 – 348.

[31] LVECZKY P C, THORVALDSEN S. Formal modeling and analysis of the OGDC wireless sensor network algorithm in real-time maude[C] // Formal Methods for Open Object-Based Distributed Systems, Ifip Wg 6.1 International Conference, Fmoods 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings. 2007 : 122 – 140.

[32] CLAVEL M, DURáN F, EKER S, et al. Maude Manual (Version 2.4)[J]. University of Illinois, 2008.

[33] CLAVEL M, DURáN F, EKER S, et al. Maude: specification and programming in rewriting logic □[J]. Theoretical Computer Science, 2002, 285(2): 187–243.

[34] ECKHARDT J, MüHLBAUER T, MESEGUER J, et al. Semantics, distributed implementation, and formal analysis of KLAIM models in Maude[J]. Science of Computer Programming, 2015, 99: 24–74.

[35] EKER S, MESEGUER J, SRIDHARANARAYANAN A. The Maude LTL Model Checker[J]. Electronic Notes in Theoretical Computer Science, 2002, 71(05): 162–187.

[36] 张民. 基于重写逻辑的 SNP 系统模型检测 [D]. [S.l.]: 上海交通大学, 2007.

[37] 姬国珍. 基于 Maude 的安全协议的形式化分析 [D]. [S.l.]: 西安电子科技大学, 2011.

[38] 尹剑飞, 王学斌. 模型转换的重写逻辑构架研究 [J]. 计算机工程与应用, 2006, 42(2): 14–16.

[39] 冯曙光. 基于 Real-Time Maude 的 AADL Thread 构件的形式化建模与验证 [D]. [S.l.]: 华东师范大学, 2016.

[40] MESEGUER J. Membership algebra as a logical framework for equational specification[J]. Lecture Notes in Computer Science, 1997, 1376: 18–61.

[41] VERDEJO A, MARTí-OLIET N. Two Case Studies of Semantics Execution in Maude: CCS and LOTOS[J]. Formal Methods in System Design, 2005, 27(1-2): 113–172.

[42] 栾天骄, 陈仪香, 王江涛. 实时系统规范语言 STeC 的 Maude 重写系统 [J]. 计算机工程, 2013, 39(10): 57–62.

[43] 胡涛. 基于 PKM 加密算法的 WiMAX 安全机制研究 [D]. [S.l.]: 上海交通大学, 2008.

[44] CROW B P, WIDJAJA I, KIM L G, et al. IEEE 802.11 Wireless Local Area Networks[J]. IEEE Communications Magazine, 1997, 35(9): 116–126.

[45] MONTENEGRO G, KUSHALNAGAR N, HUI J, et al. Transmission of IPv6 packets over IEEE 802.15. 4 networks[R]. 2007.

[46] 魏笑笑, 周政华. WIMAX 技术及其应用分析 [J]. 通信技术, 2009, 42(3): 35–38.

[47] 苏鹏, 彭建华. WIMAX 技术应用前景分析 [J]. 通信技术, 2007, 40(10): 34–36.

[48] 晋晶. 多射频多信道无线 Mesh 网络的跨层设计 [D]. [S.l.]: 北京邮电大学, 2010.

[49] 王旭. 基于 IEEE802.16d 的 WiMAX 基站系统实现技术研究 [D]. [S.l.]: 东南大学, 2007.

[50] 王鑫娜. IEEE802.16 中基于 OFDMA 的上行链路分组调度算法研究 [D]. [S.l.]: 北京邮电大学, 2006.

[51] 殷韵. IEEE802.16 协议带宽调度架构的研究 [D]. [S.l.]: 重庆大学, 2009.

[52] 杨海明. WiMAX 无线网络测试及规划 [D]. [S.l.]: 北京邮电大学, 2009.

[53] 王欣. WIMAX 系统基站 MAC 层研究与实现 [D]. [S.l.]: 西安电子科技大学, 2012.

[54] 付安民. WiMAX 无线网络中的密钥管理协议研究 [D]. [S.l.]: 西安电子科技大学, 2011.

[55] 冯伟. 基于 AES-CCM 模式的 IPSec 应用及其性能研究 [D]. [S.l.]: 西南农业大学西南大学, 2004.

[56] 王彦田. WIMAX 无线城域网安全认证研究与实现 [D]. [S.l.]: 哈尔滨理工大学, 2008.

[57] PATHAK S. Next Generation 4G WiMAX Networks - IEEE 802.16 Standard[J], 2012, 2(4): 507–518.

[58] LIU C L, LAYLAND J W. Scheduling algorithms for multiprogramming in a hard-real-time environment[J]. Journal of the ACM (JACM), 1973, 20(1): 46–61.

[59] ÖLVECZKY P C. Real-Time Maude 2.3 manual[J]. Research Report, 2004, 174: 65–81.

[60] BAE K, ESCOBAR S, MESEGUER J. The Maude LTL LBMC Tool Tutorial[J]. Formal.cs.illinois.edu, .

致　　谢

在华师大三年的学习生涯即将结束，研究生期间的经历让我着实受益良多。在学习方面，各位授课老师为我们传道受业解惑，让我学习了软件工程专业诸多专业知识，而科研以及课题实验过程则培养了我独立思考以及解决实际问题的动手能力。感谢学校提供的具有浓厚学术氛围以及温馨的生活环境。

论文的研究工作是在张民老师的指导下完成的。张民老师从课题的选择，形式化建模与验证方法，以及论文的撰写等各个方面给予我很多指导。特别是在一次次的讨论班以及课余时间，与我讨论并纠正形式化建模语言 Maude 的语法语义以及建模方法，并教导我们需具有对科研的热情以及正确的科研方法。在此向张民老师表示衷心的谢意。同时也要感谢曾经的张立臣老师给予我所有的帮助。

在理科楼 B1112 的学习生活非常丰富多彩。感谢实验室每一位同学与我和谐共处，并总能在生活与学习中伸出援助之手。感谢我的师兄师弟们，平日与我共同探讨学术问题，并在实验建模以及论文撰写中给我提供了很大的帮助。同样要感谢我的室友，与我共同经历研究生三年的种种，互相督促进步，关心与照顾。

感谢我的父母与所有的亲人，一直以来尊重我的选择，并总在我艰难困苦的时刻默默支持。感谢这一生有你们的陪伴，希望我能有更多的时间关心和照顾你们。

再次感谢每个帮助和关心我的人。

佘 葭

二零一七年四月

攻读硕士学位期间发表论文和科研情况

■ 已公开发表论文

[1] 佘葭, 张民. 基于重写逻辑的 PKMv3 协议形式化建模与验证 [J]. 计算机应用与软件, 2017.10.

[2] She J, Zhu X, Zhang M, et al. Algebraic Formalization and Verification of PKMv3 Protocol using Maude[C]// SEKE. 2017.