PacketCable™ 1.5 Specifications

Security

PKT-SP-SEC1.5-I03-090624

ISSUED

Notice

Document Status Sheet

Document Control Number:	PKT-SP-SEC1.5-I03-090624			
Document Title:	Security			
Revision History:	I01 - Issued January 28, 2005 I02 - Issued April 12, 2007 I03 - Issued June 24, 2009			
Date:	June 24, 2009			
Status:	~~Work in Progress~~	~~Draft~~	Issued	~~Closed~~
Distribution Restrictions:	~~Author Only~~	~~CL/Member/ IPR Vendor~~	~~CL/Member/ NDA Vendor~~	Public

Key to Document Status Codes:

Work in Progress	An incomplete document designed to guide discussion and generate feedback that may include several alternative requirements for consideration.
Draft	Documents in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Issued	A stable document, reviewed, tested and validated, suitable to enable cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Contents

1 SCOPE AND INTRODUCTION 9

1.1 PURPOSE 9
1.2 SCOPE 9
1.2.1 Goals 10
1.2.2 Assumptions 10
1.2.3 Requirements 11
1.3 SPECIFICATION LANGUAGE 11
1.4 DOCUMENT OVERVIEW 12

2 REFERENCES 13

2.1 NORMATIVE REFERENCES 13
2.2 INFORMATIVE REFERENCES 15
2.3 REFERENCE ACQUISITION 15

3 TERMS AND DEFINITIONS 16

4 ABBREVIATIONS AND ACRONYMS 20

5 ARCHITECTURAL OVERVIEW OF PACKETCABLE SECURITY 27

5.1 PACKETCABLE REFERENCE ARCHITECTURE 27
5.1.1 HFC Network 27
5.1.2 Call Management Server 27
5.1.3 Functional Categories 28
5.2 THREATS 30
5.2.1 Theft of Network Services 32
5.2.2 Bearer Channel Information Threats 33
5.2.3 Signaling Channel Information Threats 33
5.2.4 Service Disruption Threats 34
5.2.5 Repudiation 34
5.2.6 Threat Summary 35
5.3 SECURITY ARCHITECTURE 36
5.3.1 Overview of Security Interfaces 36
5.3.2 Security Assumptions 39
5.3.3 Susceptibility of Network Elements to Attack 41

6 SECURITY MECHANISMS 45

6.1 IPSEC 45
6.1.1 Overview 45
6.1.2 PacketCable Profile for IPsec ESP (Transport Mode) 45
6.2 INTERNET KEY EXCHANGE (IKE) 47
6.2.1 Overview 47
6.2.2 PacketCable Profile for IKE 47
6.3 SNMPV3 49
6.3.1 SNMPv3 Transform Identifiers 49
6.3.2 SNMPv3 Authentication Algorithms 49
6.4 KERBEROS / PKINIT 50
6.4.1 Overview 50
6.4.2 PKINIT Exchange 52
6.4.3 Symmetric Key AS Request / AS Reply Exchange 60
6.4.4 Kerberos TGS Request / TGS Reply Exchange 62
6.4.5 Kerberos Server Locations and Naming Conventions 65
6.4.6 MTA Principal Names 68
6.4.7 Mapping of MTA MAC Address to MTA FQDN 68
6.4.8 Server Key Management Time Out Procedure 72

6.4.9 Service Key Versioning ... *74*
6.5 KERBERIZED KEY MANAGEMENT ... 74
6.5.1 Overview ... *74*
6.5.2 Kerberized Key Management Messages ... *75*
6.5.3 Kerberized IPsec ... *85*
6.5.4 Kerberized SNMPv3 ... *89*
6.6 END-TO-END SECURITY FOR RTP ... 93
6.7 END-TO-END SECURITY FOR RTCP ... 93
6.8 BPI+ ... 94
6.9 TLS ... 95
6.9.1 Overview ... *95*
6.9.2 PacketCable Profile for TLS with SIP ... *95*

7 SECURITY PROFILE ... 97

7.1 DEVICE AND SERVICE PROVISIONING ... 98
7.1.1 Device Provisioning ... *101*
7.1.2 Subscriber Enrollment ... *108*
7.2 QUALITY OF SERVICE (QOS) SIGNALING ... 109
7.2.1 Dynamic Quality of Service (DQoS) ... *109*
7.3 BILLING SYSTEM INTERFACES ... 111
7.3.1 Security Services ... *111*
7.3.2 Cryptographic Mechanisms ... *111*
7.3.3 Key Management ... *112*
7.3.4 Billing System Summary Security Profile Matrix ... *113*
7.4 CALL SIGNALING ... 113
7.4.1 Network Call Signaling (NCS) ... *113*
7.4.2 Call Signaling Security Profile Matrix ... *119*
7.5 PSTN GATEWAY INTERFACE ... 120
7.5.1 Reference Architecture ... *120*
7.5.2 Security Services ... *120*
7.5.3 Cryptographic Mechanisms ... *121*
7.5.4 Key Management ... *121*
7.5.5 MGC-MG Summary Security Profile Matrix ... *121*
7.6 MEDIA STREAM ... 121
7.6.1 Security Services ... *121*
7.6.2 Cryptographic Mechanisms ... *122*
7.7 AUDIO SERVER SERVICES ... 141
7.7.1 Reference Architecture ... *141*
7.7.2 Security Services ... *142*
7.7.3 Cryptographic Mechanisms ... *143*
7.7.4 Key Management ... *143*
7.7.5 MPC-MP Summary Security Profile Matrix ... *144*
7.8 ELECTRONIC SURVEILLANCE INTERFACES ... 144
7.8.1 Reference Architecture ... *144*
7.8.2 Security Services ... *145*
7.8.3 Cryptographic Mechanisms ... *146*
7.8.4 Key Management ... *146*
7.8.5 Electronic Surveillance Security Profile Matrix ... *148*
7.9 CMS PROVISIONING ... 148
7.9.1 Reference Architecture ... *148*
7.9.2 Security Services ... *148*
7.9.3 Cryptographic Mechanisms ... *148*
7.9.4 Key Management ... *149*
7.9.5 Provisioning Server-CMS Summary Security Profile Matrix ... *149*

8 PACKETCABLE CERTIFICATES ... 150

8.1 GENERIC STRUCTURE 150
8.1.1 Version 150
8.1.2 Public Key Type 150
8.1.3 Extensions 150
8.1.4 Signature Algorithm 150
8.1.5 SubjectName and IssuerName 150
8.1.6 Certificate Profile Notation 151
8.2 CERTIFICATE TRUST HIERARCHY 151
8.2.1 Certificate Validation 151
8.2.2 MTA Device Certificate Hierarchy 152
8.2.3 CableLabs Service Provider Certificate Hierarchy 155
8.2.4 Certificate Revocation 163

9 CRYPTOGRAPHIC ALGORITHMS 164

9.1 AES 164
9.2 DES 164
9.2.1 XDESX 164
9.2.2 DES-CBC-PAD 164
9.2.3 3DES-EDE 164
9.3 BLOCK TERMINATION 165
9.4 RSA SIGNATURE 170
9.5 HMAC-SHA1 170
9.6 KEY DERIVATION 170
9.7 THE MMH-MAC 170
9.7.1 The MMH Function 170
9.7.2 The MMH-MAC 172
9.8 RANDOM NUMBER GENERATION 172

10 PHYSICAL SECURITY 173

10.1 PROTECTION FOR MTA KEY STORAGE 173
10.2 MTA KEY ENCAPSULATION 175

11 SECURE SOFTWARE DOWNLOAD 176

APPENDIX A. PACKETCABLE ADMIN GUIDELINES & BEST PRACTICES (INFORMATIVE) 177

APPENDIX B. KERBEROS NETWORK AUTHENTICATION SERVICE (NORMATIVE). 178

APPENDIX C. PKINIT SPECIFICATION 271

APPENDIX D. EXAMPLE OF MMH ALGORITHM IMPLEMENTATION (INFORMATIVE) 290

APPENDIX E. OAKLEY GROUPS 295

APPENDIX F. ACKNOWLEDGEMENTS 297

APPENDIX G. REVISION HISTORY 298

List of Figures

FIGURE 1. PACKETCABLE SINGLE ZONE ARCHITECTURE 27
FIGURE 2. PACKETCABLE SECURED INTERFACES 31
FIGURE 3. PACKETCABLE SECURITY INTERFACES WITH KEY-MANAGEMENT 37
FIGURE 4. KERBEROS-BASED KEY MANAGEMENT FOR IPSEC 51
FIGURE 5. PKINIT EXCHANGE 53
FIGURE 6. SYMMETRIC-KEY AS REQUEST / AS REPLY EXCHANGE 61
FIGURE 7. KERBEROS TGS REQUEST / TGS REPLY EXCHANGE 63
FIGURE 8. KERBEROS AP REQUEST / AP REPLY EXCHANGE 75
FIGURE 9. REKEY MESSAGE TO ESTABLISH A SECURITY PARAMETER 80
FIGURE 10. PACKETCABLE PROVISIONING FLOWS 99
FIGURE 11. QOS SIGNALING INTERFACES IN PACKETCABLE NETWORK 109
FIGURE 12. NCS REFERENCE ARCHITECTURE 114
FIGURE 13. KEY MANAGEMENT FOR NCS CLUSTERS 116
FIGURE 14. CMS – CMS SIGNALING FLOW WITH SECURITY 119
FIGURE 15. RTP PACKET HEADER FORMAT 123
FIGURE 16. FORMAT OF ENCODED RTP PACKET 123
FIGURE 17. RTP PACKET PROFILE CHARACTERISTICS 125
FIGURE 18. RTCP PACKET FORMAT 128
FIGURE 19. RTCP ENCRYPTED PACKET FORMAT 128
FIGURE 20. END-END SECRET DISTRIBUTION OVER NCS 131
FIGURE 21. AUDIO SERVER COMPONENTS AND INTERFACES 142
FIGURE 22. ELECTRONIC SURVEILLANCE SECURITY INTERFACES 145
FIGURE 23. PACKETCABLE CERTIFICATE HIERARCHY 151
FIGURE 24. CBC MODE 165
FIGURE 25. CBC PAD MODE 166
FIGURE 26. DESX-XEX AS BLOCK CIPHER 167
FIGURE 27. 3DES-EDE AS BLOCK CIPHER 168
FIGURE 28. CBC WITH RESIDUAL BLOCK TERMINATION 169

List of Tables

TABLE 1. PACKETCABLE SECURITY INTERFACES TABLE 38
TABLE 2. IPSEC ESP TRANSFORM IDENTIFIERS 46
TABLE 3. IPSEC AUTHENTICATION ALGORITHMS 46
TABLE 4. SNMPV3 TRANSFORM IDENTIFIERS 49
TABLE 5. SNMPV3 AUTHENTICATION ALGORITHMS 49
TABLE 6. MTA FQDN REQUEST FORMAT 69
TABLE 7. KRB_SAFE FORMAT 70
TABLE 8. MTA FQDN FORMAT 70
TABLE 9. KRB_SAFE DATA FORMAT 71
TABLE 10. MAPPING OF KRB_MTAMAP_ERR TO KRB_ERR 72
TABLE 11. EXAMPLE IPSEC SECURITY POLICY DATABASE ENTRIES FOR NCS SIGNALING BETWEEN MTA AND CMS 85
TABLE 12. REQUIRED FORMAT FOR DATA IN THE AP REQUEST 90
TABLE 13. REQUIRED FORMAT FOR DATA IN THE AP REPLY 90
TABLE 14. RTP PACKET TRANSFORM IDENTIFIERS 93
TABLE 15. RTP PACKETCABLE AUTHENTICATION ALGORITHMS 93
TABLE 16. RTCP PACKET TRANSFORM IDENTIFIERS 94
TABLE 17. RTCP AUTHENTICATION ALGORITHMS 94
TABLE 18. TLS CIPHERSUITES 95
TABLE 19. RTP – RTCP SECURITY PROFILE MATRIX 97
TABLE 20. KERBEROS KEY MANAGEMENT DURING MTA PROVISIONING 100
TABLE 21. POST-MTA PROVISIONING SECURITY FLOWS 104
TABLE 22. SECURITY PROFILE MATRIX – MTA DEVICE PROVISIONING 108
TABLE 23. SECURITY PROFILE MATRIX – DQOS 111
TABLE 24. SECURITY PROFILE MATRIX – RADIUS 113
TABLE 25. SECURITY PROFILE MATRIX – NETWORK CALL SIGNALING 119
TABLE 26. SECURITY PROFILE MATRIX – TGCP 121
TABLE 27. SECURITY PROFILE MATRIX – RTP & RTCP 141
TABLE 28. SECURITY PROFILE MATRIX – AUDIO SERVER SERVICES 144
TABLE 29. SECURITY PROFILE MATRIX – ELECTRONIC SURVEILLANCE 148
TABLE 30. SECURITY PROFILE MATRIX – CMS PROVISIONING 149
TABLE 31. MTA ROOT CERTIFICATE 153
TABLE 32. MTA MANUFACTURER CERTIFICATE 154
TABLE 33. MTA DEVICE CERTIFICATE 155
TABLE 34. CABLELABS SERVICE PROVIDER ROOT CERTIFICATE 156
TABLE 35. SERVICE PROVIDER CA CERTIFICATE 157
TABLE 36. LOCAL SYSTEM CA CERTIFICATE 158
TABLE 37. KEY DISTRIBUTION CENTER CERTIFICATE 159
TABLE 38. DF CERTIFICATE 159
TABLE 39. PACKETCABLE SERVER CERTIFICATES 161
TABLE 40. PACKETCABLE TLS CERTIFICATES 162

This page intentionally left blank.

1 SCOPE AND INTRODUCTION

1.1 Purpose

PacketCable™, a project conducted by Cable Television Laboratories, Inc. (CableLabs®) and its member companies, is aimed at identifying, qualifying, and supporting packet-based voice and video products over cable systems. These products represent new classes of services utilizing cable-based packet communication networks. New service classes in the near term include voice communications and videoconferencing over cable networks and the Internet.

PacketCable is a set of protocols and associated element functional requirements developed to provide the capability to deliver Quality-of-Service (QoS) enhanced secure communications services using packetized data transmission technology to a consumer's home over the cable television Hybrid Fiber/Coax (HFC) data network. PacketCable utilizes a network superstructure that overlays the two-way data-ready cable television network. While the initial service offerings in the PacketCable product line are anticipated to be Packet Voice and Packet Video, the long-term project vision encompasses a large family of packet-based services.

The purpose of any security technology is to protect items of value, whether a revenue stream, or a purchasable information asset of some type. Threats to this revenue stream exist when a user of the network perceives the value, expends effort and money, and invents a technique to get around the necessary payments. Some network users will go to extreme lengths to steal when they perceive extreme value. The addition of security technology to protect value has an associated cost; the more expended, the more secure one can be. The proper engineering task is to employ a reasonable costing security technology to force any user with the intent to steal or disrupt network services to spend an unreasonable amount of money to circumvent it. Security effectiveness is thus basic economics.

In addition, a PacketCable network used to offer voice communications must be at least as secure as the Public Switched Telephone Network (PSTN) networks are today. Much of the PSTN security depends on the fact that each telephone is connected to a dedicated line. In order to provide the same level of privacy and resistance to denial-of-service attacks when a PacketCable IP network is used for voice communications, appropriate cryptography-based security mechanisms have been specified. This secures both voice and signaling data transmitted over a shared HFC network and over a shared IP backbone.

1.2 Scope

The scope of this document is to define the PacketCable Security architecture, protocols, algorithms, associated functional requirements and any technological requirements that can provide for the security of the system for the PacketCable network. Authentication, access control, signaling and media content integrity, confidentiality, and non-repudiation security services must be provided as defined herein for each of the network element interfaces.

PacketCable security spans the entire PacketCable architecture. The PacketCable Architecture 1.5 Technical Report [1] defines the overall PacketCable architecture, as well as the system elements, interfaces, and functional requirements for the entire PacketCable network. These and all PacketCable specifications can be found at www.packetcable.com.

From time to time this document refers to the voice communications capabilities of a PacketCable network in terms of "IP Telephony." The legal/regulatory classification of IP-based voice communications provided over cable networks and otherwise, and the legal/regulatory obligations, if any, borne by providers of such voice communications, are not yet fully defined by appropriate legal and regulatory authorities. Nothing in this specification is addressed to, or intended to affect, those issues.

In particular, while this document uses standard terms such as "call," "call signaling," telephony," etc., it should be recalled that while a PacketCable network performs activities analogous to these PSTN functions, the manner by which it does so differs considerably from the manner in which they are performed in the PSTN by telecommunications carriers, and that these differences may be significant for legal/regulatory purposes. Moreover, while reference is made here to "IP Telephony," it should be recognized that this term embraces a number of different technologies and network architectures, each with

different potential associated legal/regulatory obligations. No particular legal/regulatory consequences are assumed or implied by the use of this term. This specification makes use of existing standards wherever possible. Whenever there is an existing standard used in the definition of any requirement in this specification, the related existing standard will be referenced. When there are options defined with respect to the existing standards, this specification will explicitly define the options within the existing standard that are supported.

1.2.1 Goals

This specification describes the security relationships between the elements on the PacketCable network. The general goals of the PacketCable network security specification and any implementations that encompass the requirements defined herein should be:

- **Secure network communications** The PacketCable network security must define a security architecture, methods, algorithms and protocols that meet the stated security service requirement. All media packets and all sensitive signaling communication across the network must be safe from eavesdropping. Unauthorized message modification, insertion, deletion and replays anywhere in the network must be easily detectable and must not affect proper network operation.
- **Reasonable cost** The PacketCable network security must define security methods, algorithms and protocols that meet the stated security service requirements such that a reasonable implementation can be manifested with reasonable cost and implementation complexity.
- **Network element interoperability** All of the security services for any of the PacketCable network elements must inter-operate with the security services for all of the other PacketCable network elements. Multiple vendors may implement each of the PacketCable network elements as well as multiple vendors for a single PacketCable network element.
- **Extensibility** The PacketCable security architecture, methods, algorithms and protocols must provide a framework into which new security methods and algorithms may be incorporated as necessary.

1.2.2 Assumptions

The following assumptions are made relative to the current scope of the PacketCable Security Specification:

- Embedded Multimedia Terminal Adapters (E-MTAs) and Standalone Multimedia Terminal Adapters (S-MTAs) are within the scope of this specification.
- NCS is the only call signaling method, on the access network, addressed in this specification.
- This version of the PacketCable Security Specification specifies security for a single administrative domain and the communications between domains.
- Security for chained RADIUS servers is not currently in the scope.
- The PacketCable Security Specification does not have a requirement for exportability outside the United States; exportability of encryption algorithms is not addressed in this specification.
- This specification also does not include requirements for associated security operational issues (e.g., site security), back-office or inter/intra back-office security, service authorization policies or secure database handling. Record Keeping Servers (RKS), Network Management Systems, File Transfer Protocol (FTP) servers and Dynamic Host Configuration Protocol (DHCP) servers are all considered to be unique to any service provider's implementation and are beyond the scope of this specification.
- This specification assumes that MTAs implement the specified PacketCable 1.5 modules, and optionally, can implement the IETF IPCDN MIB modules. As such, any reference to a specific

MIB Object is assumed to be a reference in either MIB module unless explicitly specified or the MIB Object exists in only one set of MIB modules.

1.2.3 Requirements

The following requirement is made relative to the current scope of the PacketCable Security Specification:

- All E-MTAs must use DOCSIS™ (1.1 or later)-compliant cable modems and implement BPI+ [9]. Any references to DOCSIS, including specific references to DOCSIS versions 1.1 or 2.0, are understood to refer to DOCSIS version 1.1 or later.

1.3 Specification Language

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST"	This word or the adjective "REQUIRED" means that the item is an absolute requirement of this specification.
"MUST NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or event useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
"MAY"	This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

The legal/regulatory classification of IP-based voice communications provided over cable networks and otherwise, and the legal/regulatory obligations, if any, borne by providers of such voice communications, are not yet fully defined by appropriate legal and regulatory authorities. Nothing in this specification is addressed to, or intended to affect, those issues. In particular, while this document uses standard terms such as "call," "call signaling," "telephony," etc., it will be evident from this document that while a Packet-Cable network performs activities analogous to these PSTN functions, the manner by which it does so differs considerably from the manner in which they are performed in the PSTN by telecommunications carriers. These differences may be significant for legal/regulatory purposes.

1.4 Document Overview

This specification covers security for the entire PacketCable architecture. This specification describes the PacketCable architecture, identifies security risks and specifies mechanisms to secure the architecture. The document is structured as follows:

- Architectural Overview of PacketCable. The initial section describes the PacketCable architecture as a point of reference for the remainder of the document. Refer to the PacketCable 1.5 Architecture Framework Technical Report [1] and each individual specification for full details.
- Security Threats are described in the context of the reference architecture.
- The overall security architecture and security assumptions are described.
- Security Mechanisms. This section specifies how public domain security mechanisms are to be implemented in PacketCable including IPsec, Internet Key Exchange (IKE), Kerberos with PKINIT, media stream security, BPI+ and RADIUS.
- Security Profile. This section profiles the security for each major area of the PacketCable architecture. The profile includes a description of the security requirements as well as the specifications for securing at-risk interfaces. Refer to the individual specifications for details about each PacketCable interface.
- PacketCable X.509 Certificate Profile and Management. X.509 Certificates are specified for a number of devices and functions within the PacketCable architecture. This section describes the format of the Certificates as well as the trust hierarchy for Certificate management within PacketCable.
- Cryptographic Algorithms. This section specifies the details of cryptographic algorithms specified in the PacketCable security architecture.
- Physical Security. This section documents assumptions about the physical security of the MTA keys.
- Secure Software Download. This section specifies the secure loading and upgrading of software to the MTAs.

2 REFERENCES

2.1 Normative References

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Notwithstanding, intellectual property rights may be required to use or implement such normative references.

[1] PacketCable 1.5 Architecture Framework Technical Report, PKT-TR-ARCH1.5-V02-070412, April 12, 2007, Cable Television Laboratories, Inc.

[2] PacketCable 1.5 Network-Based Call Signaling Protocol Specification, PKT-SP-NCS1.5-I03-070412, April 12, 2007, Cable Television Laboratories, Inc.

[3] PacketCable 1.5 Dynamic Quality of Service Specification, PKT-SP-DQOS1.5-I04-090624, June 24, 2009, Cable Television Laboratories, Inc.

[4] PacketCable 1.5 MTA Device Provisioning Specification, PKT-SP-PROV1.5-I04-090624, June 24, 2009, Cable Television Laboratories, Inc.

[5] PacketCable 1.5 PSTN Gateway Call Signaling Protocol Specification, PKT-SP-TGCP1.5-I03-070412, April 12, 2007, Cable Television Laboratories, Inc.

[6] PacketCable 1.5 Event Messages, PKT-SP-EM1.5-I03-070412, April 12, 2007, Cable Television Laboratories, Inc.

[7] PacketCable 1.5 Audio/Video Codecs Specification, PKT-SP-CODEC1.5-I03-090624, June 24, 2009, Cable Television Laboratories, Inc.

[8] Data-Over-Cable Service Interface Specifications, Radio Frequency Interface Specification, SP-RFIv1.1-C01-050907, September 7, 2005, Cable Television Laboratories, Inc.

[9] Data-Over-Cable Service Interface Specifications, Baseline Privacy Plus Interface Specification, SP-BPI+-C01-081104, November 4, 2008, Cable Television Laboratories, Inc.

[10] RFC 1889, RTP: A Transport Protocol for Real-Time Applications, January, 1996.

[11] IETF RFC 2104, HMAC: Keyed-Hashing for Message Authentication, February 1997.

[12] IETF RFC 2630, Cryptographic Message Syntax, June 1999.

[13] IETF RFC 2866, RADIUS Accounting, June 2000.

[14] IETF RFC 2327, SDP: Session Description Protocol, April 1998.

[15] NIST, FIPS 180-1, Secure Hash Algorithm, Department of Commerce, April 1995.

[16] IETF RFC 2437, PKCS#1: RSA Cryptography Specifications Version 2.0, October 1998.

[17] IETF RFC 2246, The TLS Protocol Version 1.0, January 1999.

[18] S. Halevi and H. Krawczyk, MMH: Software Message Authentication in Gbit/sec Rates," Proceedings of the 4th Workshop on Fast Software Encryption, (1997) vol. 1267 Springer-Verlag, pp. 172-189.

[19] IETF RFC 2401, Security Architecture for the Internet Protocol, November 1998.

[20] IETF RFC 2406, IP Encapsulating Security Payload (ESP), November 1998.

[21] IETF RFC 2407, The Internet IP Security Domain of Interpretation for ISAKMP, November 1998.

[22] IETF RFC 2451, The ESP CBC-Mode Cipher Algorithms, November 1998.

[23] IETF RFC 2404, The Use of HMAC-SHA-1-96 within ESP and AH, November 1998.

[24] IETF RFC 2409, The Internet Key Exchange (IKE), November 1998.

[25] PacketCable 1.5 MTA MIB, PKT-SP-MIB-MTA1.5-I01-050128, January 28, 2005, Cable Television Laboratories, Inc.

[26] PacketCable 1.5 Signaling MIB, PKT-SP-MIB-SIG1.5-I01-050128, January 28, 2005, Cable Television Laboratories, Inc.

[27] PacketCable 1.5 Audio Server Protocol, PKT-SP-ASP1.5-I02-070412, April 12, 2007, Cable Television Laboratories, Inc.

[28] IETF RFC 3414, User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), December 2002.

[29] IETF RFC 2367, PF_KEY Key Management API, Version 2, July 1998.

[30] PacketCable 1.5 Electronic Surveillance Specification, PKT-SP-ESP1.5-I02-070412, April 12, 2007, Cable Television Laboratories, Inc.

[31] FIPS-81, Federal Information Processing Standards Publication DES Modes of Operation, December 1980.

[32] PacketCable 1.5 Call Management Server Signaling Specification, PKT-SP-CMSS1.5-I05-090624, June 24, 2009, Cable Television Laboratories, Inc.

[33] ITU-T Recommendation X.509 (1997 E): Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, June 1997.

[34] IETF RFC 2459, Internet X.509 Public Key Infrastructure Certificate and CRL Profile, January 1999.

[35] FIPS197 Advanced Encryption Standard (AES), Department of Commerce, November 26, 2001.

[36] ITU-T Recommendation X.690, Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), December, 1997.

[37] IETF RFC 2403, The Use of HMAC-MD5-96 within ESP and AH, November, 1998.

[38] IETF RFC 3412, Message Processing and Dispatching for the Simple Network Management Protocol (SNMP), December 2002.

[39] IETF RFC 2782, A DNS RR for specifying the location of services (DNS SRV), February 2000.

[40] IETF RFC 3261, SIP: Session Initiation Protocol, February 2002.

[41] IETF RFC 3268, AES Ciphersuites for TLS, June 2002.

[42] IETF RFC 1890, RTP Profile for Audio and Video Conferences with Minimal Control, January 1996.

2.2 Informative References

[43] Applied Cryptography, B. Schneier, John Wiley & Sons Inc, second edition, 1996.

[44] IETF RFC 1750, Randomness Recommendations for Security, December 1994.

[45] How to Protect DES Against Exhaustive Key Search, J. Killian, P. Rogaway, (Edited version presented at Proceedings of Crypto '96), July 1997.

[46] FIPS 140-2, Federal Information Processing Standards Publication (FIPS PUB) 140-2, Security Requirements for Cryptographic Modules, May 2000.

[47] IETF RFC 4682, Multimedia Terminal Adapter (MTA) Management Information Base for PacketCable- and IPCablecom-Compliant Devices. December 2006.

[48] IETF RFC 5098 Signaling MIB for PacketCable and IPCablecom Multimedia Terminal Adapters (MTAs), February 2008

2.3 Reference Acquisition

- Cable Television Laboratories, Inc., 858 Coal Creek Circle, Louisville, CO 80027; Phone 303-661-9100; Fax 303-661-9199; Internet: http://www.cablelabs.com
- ITU-T Recommendations available at http://www.itu.int
- IETF RFCs available at http://www.ietf.org/rfc.html
- FIPS publications available at http://www.itl.nist.gov/fipspubs/

3 TERMS AND DEFINITIONS

The PacketCable suite of documents use the following terms:

Access Control	Limiting the flow of information from the resources of a system only to authorized persons, programs, processes, or other system resources on a network.
Active	A service flow is said to be "active" when it is permitted to forward data packets. A service flow must first be admitted before it is active.
Admitted	A service flow is said to be "admitted" when the CMTS has reserved resources (e.g., bandwidth) for it on the DOCSIS™ network.
A-link	A-Links are SS7 links that interconnect STPs and either SSPs or SCPs. 'A' stands for "Access."
Asymmetric Key	An encryption key or a decryption key used in public key cryptography, where encryption and decryption keys are always distinct.
Audio Server	An Audio Server plays informational announcements in PacketCable network. Media announcements are needed for communications that do not complete and to provide enhanced information services to the user. The component parts of Audio Server services are Media Players and Media Player Controllers.
Authentication	The process of verifying the claimed identity of an entity to another entity.
Authenticity	The ability to ensure that the given information is without modification or forgery and was in fact produced by the entity that claims to have given the information.
Authorization	The act of giving access to a service or device if one has permission to have the access.
Cipher	An algorithm that transforms data between plaintext and ciphertext.
Ciphersuite	A set which must contain both an encryption algorithm and a message authentication algorithm (e.g., a MAC or an HMAC). In general, it may also contain a key-management algorithm, which does not apply in the context of PacketCable.
Ciphertext	The (encrypted) message output from a cryptographic algorithm that is in a format that is unintelligible.
Cleartext	The original (unencrypted) state of a message or data. Also called plaintext.
Confidentiality	A way to ensure that information is not disclosed to anyone other then the intended parties. Information is encrypted to provide confidentiality. Also known as privacy.
Cryptanalysis	The process of recovering the plaintext of a message or the encryption key without access to the key.
Cryptographic algorithm	An algorithm used to transfer text between plaintext and ciphertext.
Decipherment	A procedure applied to ciphertext to translate it into plaintext.
Decryption	A procedure applied to ciphertext to translate it into plaintext.
Decryption key	The key in the cryptographic algorithm to translate the ciphertext to plaintext.
Digital certificate	A binding between an entity's public key and one or more attributes relating to its identity, also known as a public key certificate.

Digital signature	A data value generated by a public-key algorithm based on the contents of a block of data and a private key, yielding an individualized cryptographic checksum.
Downstream	The direction from the headend toward the subscriber location.
Encipherment	A method used to translate plaintext into ciphertext.
Encryption	A method used to translate plaintext into ciphertext.
Encryption Key	The key used in a cryptographic algorithm to translate the plaintext to ciphertext.
Endpoint	A Terminal, Gateway or Multipoint Conference Unit (MCU).
Errored Second	Any 1-second interval containing at least one bit error.
Event Message	A message capturing a single portion of a connection.
F-link	F-Links are SS7 links that directly connect two SS7 end points, such as two SSPs. ‘F’ stands for "Fully Associated."
Flow [DOCSIS Flow]	(a.k.a. DOCSIS-QoS "service flow") A unidirectional sequence of packets associated with a Service ID (SID) and a QoS. Multiple multimedia streams may be carried in a single DOCSIS Flow.
Flow [IP Flow]	A unidirectional sequence of packets identified by OSI Layer 3 and Layer 4 header information. This information includes source/destination IP addresses, source/destination port numbers, protocol ID. Multiple multimedia streams may be carried in a single IP Flow.
Gateway	Devices bridging between the PacketCable IP Voice Communication world and the PSTN. Examples are the Media Gateway, which provides the bearer circuit interfaces to the PSTN and transcodes the media stream, and the Signaling Gateway, which sends and receives circuit switched network signaling to the edge of the PacketCable network.
H.323	An ITU-T recommendation for transmitting and controlling audio and video information. The H.323 recommendation requires the use of the ITU-T H.225 and ITU-T H.245 protocol for communication control between a "gateway" audio/video endpoint and a "gatekeeper" function.
Header	Protocol control information located at the beginning of a protocol data unit.
Integrity	A way to ensure that information is not modified except by those who are authorized to do so.
IntraLATA	Within a Local Access Transport Area.
Jitter	Variability in the delay of a stream of incoming packets making up a flow such as a voice communication.
Kerberos	A secret-key network authentication protocol that uses a choice of cryptographic algorithms for encryption and a centralized key database for authentication.
Key	A mathematical value input into the selected cryptographic algorithm.
Key Exchange	The swapping of public keys between entities to be used to encrypt communication between the entities.
Key Management	The process of distributing shared symmetric keys needed to run a security protocol.
Key Pair	An associated public and private key where the correspondence between the two are mathematically related, but it is computationally infeasible to derive the private key from the public key.
Keying Material	A set of cryptographic keys and their associated parameters, normally associated with a particular run of a security protocol.

Keyspace	The range of all possible values of the key for a particular cryptographic algorithm.
Latency	The time, expressed in quantity of symbols, taken for a signal element to pass through a device.
Link Encryption	Cryptography applied to data as it travels on data links between the network devices.
Network Layer	Layer 3 in the Open System Interconnection (OSI) architecture that provides network information that is independent from the lower layers.
Network Management	The functions related to the management of data across the network.
Network Management OSS	The functions related to the management of data link layer and physical layer resources and their stations across the data network supported by the hybrid fiber/coax system.
Nonce	A random value used only once that is sent in a communications protocol exchange to prevent replay attacks.
Non-Repudiation	The ability to prevent a sender from denying later that he or she sent a message or performed an action.
Off-Net Call	A communication connecting a PacketCable subscriber out to a user on the PSTN.
On-Net Call	A communication placed by one customer to another customer entirely on the PacketCable Network.
One-way Hash	A hash function that has an insignificant number of collisions upon output.
Plaintext	The original (unencrypted) state of a message or data. Also called cleartext.
Pre-shared Key	A shared secret key passed to both parties in a communication flow, using an unspecified manual or out-of-band mechanism.
Privacy	A way to ensure that information is not disclosed to any one other then the intended parties. Information is usually encrypted to provide confidentiality. Also known as confidentiality.
Private Key	The key used in public key cryptography that belongs to an individual entity and must be kept secret.
Proxy	A facility that indirectly provides some service or acts as a representative in delivering information, thereby eliminating the need for a host to support the service.
Public Key	The key used in public key cryptography that belongs to an individual entity and is distributed publicly. Other entities use this key to encrypt data to be sent to the owner of the key.
Public Key Certificate	A binding between an entity's public key and one or more attributes relating to its identity, also known as a digital certificate.
Public Key Cryptography	A procedure that uses a pair of keys, a public key and a private key, for encryption and decryption, also known as an asymmetric algorithm. A user's public key is publicly available for others to use to send a message to the owner of the key. A user's private key is kept secret and is the only key that can decrypt messages sent encrypted by the user's public key.
Root Private Key	The private signing key of the highest-level Certification Authority. It is normally used to sign public key certificates for lower-level Certification Authorities or other entities.
Root Public Key	The public key of the highest level Certification Authority, normally used to verify digital signatures generated with the corresponding root private key.

Secret Key	The cryptographic key used in a symmetric key algorithm, which results in the secrecy of the encrypted data depending solely upon keeping the key a secret, also known as a symmetric key.
Session Key	A cryptographic key intended to encrypt data for a limited period of time, typically between a pair of entities.
Signed and Sealed	An "envelope" of information which has been signed with a digital signature and sealed using encryption.
Subflow	A unidirectional flow of IP packets characterized by a single source and destination IP address and single source and destination UDP/TCP port.
Symmetric Key	The cryptographic key used in a symmetric key algorithm, which results in the secrecy of the encrypted data depending solely upon keeping the key a secret, also known as a secret key.
Systems Management	Functions in the application layer related to the management of various Open Systems Interconnection (OSI) resources and their status across all layers of the OSI architecture.
Transit Delays	The time difference between the instant at which the first bit of a Protocol Data Unit (PDU) crosses one designated boundary, and the instant at which the last bit of the same PDU crosses a second designated boundary.
Trunk	An analog or digital connection from a circuit switch that carries user media content and may carry voice signaling (M_F, R_2, etc.).
Tunnel Mode	An IPsec (ESP or AH) mode that is applied to an IP tunnel, where an outer IP packet header (of an intermediate destination) is added on top of the original, inner IP header. In this case, the ESP or AH transform treats the inner IP header as if it were part of the packet payload. When the packet reaches the intermediate destination, the tunnel terminates and both the outer IP packet header and the IPsec ESP or AH transform are taken out.
Upstream	The direction from the subscriber location toward the headend.
X.509 certificate	A public key certificate specification developed as part of the ITU-T X.500 standards directory.

4 ABBREVIATIONS AND ACRONYMS

The PacketCable suite of documents use the following abbreviations and acronyms.

AAA	Authentication, Authorization and Accounting.
AES	Advanced Encryption Standard. A block cipher, used to encrypt the media traffic in PacketCable.
AF	Assured Forwarding. This is a DiffServ Per Hop Behavior.
AH	Authentication header. An IPsec security protocol that provides message integrity for complete IP packets, including the IP header.
AMA	Automated Message Accounting. A standard form of call detail records (CDRs) developed and administered by Bellcore (now Telcordia Technologies).
ASD	Application-Specific Data. A field in some Kerberos key management messages that carries information specific to the security protocol for which the keys are being negotiated.
ASP	Audio Server Protocol.
AT	Access Tandem.
ATM	Asynchronous Transfer Mode. A protocol for the transmission of a variety of digital signals using uniform 53-byte cells.
BAF	Bellcore AMA Format, also known as AMA.
BCID	Billing Correlation ID.
BPI+	Baseline Privacy Plus Interface Specification. The security portion of the DOCSIS 1.1 standard that runs on the MAC layer.
CA	Certification Authority. A trusted organization that accepts certificate applications from entities, authenticates applications, issues certificates and maintains status information about certificates.
CA	Call Agent. The part of the CMS that maintains the communication state, and controls the line side of the communication.
CBC	Cipher Block Chaining mode. An option in block ciphers that combine (XOR) the previous block of ciphertext with the current block of plaintext before encrypting that block of the message.
CBR	Constant Bit Rate.
CDR	Call Detail Record. A single CDR is generated at the end of each billable activity. A single billable activity may also generate multiple CDRs.
CIC	Circuit Identification Code. In ANSI SS7, a two-octet number that uniquely identifies a DSO circuit within the scope of a single SS7 Point Code.
CID	Circuit ID (Pronounced "kid"). This uniquely identifies an ISUP DS0 circuit on a Media Gateway. It is a combination of the circuit's SS7 gateway point code and Circuit Identification Code (CIC). The SS7 DPC is associated with the Signaling Gateway that has domain over the circuit in question.
CIF	Common Intermediate Format.
CIR	Committed Information Rate.
CM	DOCSIS Cable Modem.
CMS	Cryptographic Message Syntax.
CMS	Call Management Server. Controls the audio connections. Also called a Call Agent in MGCP/SGCP terminology. This is one example of an Application Server.

CMTS	Cable Modem Termination System. The device at a cable headend which implements the DOCSIS RFI MAC protocol and connects to CMs over an HFC network.
CMSS	Call Management Server Signaling.
Codec	COder-DECoder.
COPS	Common Open Policy Service. Defined in RFC 2748.
CoS	Class of Service. The type 4 tuple of a DOCSIS configuration file.
CRCX	Create Connection.
CSR	Customer Service Representative.
DA	Directory Assistance.
DE	Default. This is a DiffServ Per Hop Behavior.
DES	Data Encryption Standard.
DF	Delivery Function.
DHCP	Dynamic Host Configuration Protocol.
DHCP-D	DHCP Default. Network Provider DHCP Server.
DNS	Domain Name Service.
DOCSIS®	Data-Over-Cable Service Interface Specifications.
DPC	Destination Point Code. In ANSI SS7, a 3-octet number which uniquely identifies an SS7 Signaling Point, either an SSP, STP, or SCP.
DQoS	Dynamic Quality-of-Service. Assigned on the fly for each communication depending on the QoS requested.
DSA	Dynamic Service Add.
DSC	Dynamic Service Change.
DSCP	DiffServ Code Point. A field in every IP packet that identifies the DiffServ Per Hop Behavior. In IP version 4, the TOS byte is redefined to be the DSCP. In IP version 6, the Traffic Class octet is used as the DSCP.
DTMF	Dual-tone Multi Frequency (tones).
EF	Expedited Forwarding. A DiffServ Per Hop Behavior.
E-MTA	Embedded MTA. A single node that contains both an MTA and a cable modem.
EO	End Office.
ESP	IPsec Encapsulating Security Payload. Protocol that provides both IP packet encryption and optional message integrity, not covering the IP packet header.
ETSI	European Telecommunications Standards Institute.
F-link	F-Links are SS7 links that directly connect two SS7 end points, such as two SSPs. ‘F’ stands for "Fully Associated."
FEID	Financial Entity ID.
FGD	Feature Group D signaling.
FQDN	Fully Qualified Domain Name. Refer to IETF RFC 2821 for details.
GC	Gate Controller.
GTT	Global Title Translation.
HFC	Hybrid Fiber/Coaxial. An HFC system is a broadband bi-directional shared media transmission system using fiber trunks between the headend and the fiber nodes, and coaxial distribution from the fiber nodes to the customer locations.
HMAC	Hashed Message Authentication Code. A message authentication algorithm, based on either SHA-1 or MD5 hash and defined in IETF RFC 2104.

HTTP	Hypertext Transfer Protocol. Refer to IETF RFC 1945 and RFC 2068.
IANA	Internet Assigned Numbered Authority. See www.ietf.org for details.
IC	Inter-exchange Carrier.
IETF	Internet Engineering Task Force. A body responsible, among other things, for developing standards used on the Internet. See www.ietf.org for details.
IKE	Internet Key Exchange. A key-management mechanism used to negotiate and derive keys for SAs in IPsec.
IKE–	A notation defined to refer to the use of IKE with pre-shared keys for authentication.
IKE+	A notation defined to refer to the use of IKE with X.509 certificates for authentication.
IP	Internet Protocol. An Internet network-layer protocol.
IPsec	Internet Protocol Security. A collection of Internet standards for protecting IP packets with encryption and authentication.
ISDN	Integrated Services Digital Network.
ISTP	Internet Signaling Transport Protocol.
ISUP	ISDN User Part. A protocol within the SS7 suite of protocols that is used for call signaling within an SS7 network.
ITU	International Telecommunication Union.
ITU-T	International Telecommunication Union–Telecommunication Standardization Sector.
IVR	Interactive Voice Response system.
KDC	Key Distribution Center.
LATA	Local Access and Transport Area.
LD	Long Distance.
LIDB	Line Information Database. Contains customer information required for real-time access such as calling card personal identification numbers (PINs) for real-time validation.
LLC	Logical Link Control. The Ethernet Packet header and optional 802.1P tag which may encapsulate an IP packet. A sublayer of the Data Link Layer.
LNP	Local Number Portability. Allows a customer to retain the same number when switching from one local service provider to another.
LSSGR	LATA Switching Systems Generic Requirements.
MAC	Message Authentication Code. A fixed-length data item that is sent together with a message to ensure integrity, also known as a MIC.
MAC	Media Access Control. It is a sublayer of the Data Link Layer. It normally runs directly over the physical layer.
MC	Multipoint Controller.
MCU	Multipoint Conferencing Unit.
MD5	Message Digest 5. A one-way hash algorithm that maps variable length plaintext into fixed-length (16 byte) ciphertext.
MDCP	Media Device Control Protocol. A media gateway control specification submitted to IETF by Lucent. Now called SCTP.
MDCX	Modify Connection.
MDU	Multi-Dwelling Unit. Multiple units within the same physical building. The term is usually associated with high-rise buildings.
MEGACO	Media Gateway Control IETF working group. See www.ietf.org for details.
MF	Multi-Frequency.

MG	Media Gateway. Provides the bearer circuit interfaces to the PSTN and transcodes the media stream.
MGC	Media Gateway Controller. The overall controller function of the PSTN gateway. Receives, controls and mediates call-signaling information between the PacketCable and PSTN.
MGCP	Media Gateway Control Protocol. Protocol follow-on to SGCP. Refer to IETF 2705.
MIB	Management Information Base.
MIC	Message Integrity Code. A fixed-length data item that is sent together with a message to ensure integrity, also known as a Message Authentication Code (MAC).
MMC	Multi-Point Mixing Controller. A conferencing device for mixing media streams of multiple connections.
MSB	Most Significant Bit.
MSO	Multi-System Operator. A cable company that operates many headend locations in several cities.
MSU	Message Signal Unit.
MTA	Multimedia Terminal Adapter. Contains the interface to a physical voice device, a network interface, CODECs, and all signaling and encapsulation functions required for VoIP transport, class features signaling, and QoS signaling.
MTP	The Message Transfer Part. A set of two protocols (MTP 2, MTP 3) within the SS7 suite of protocols that are used to implement physical, data link, and network-level transport facilities within an SS7 network.
MWD	Maximum Waiting Delay.
NANP	North American Numbering Plan.
NANPNAT	North American Numbering Plan Network Address Translation.
NAT Network Layer	Network Address Translation. Layer 3 in the Open System Interconnection (OSI) architecture. This layer provides services to establish a path between open systems.
NCS	Network Call Signaling.
NPA-NXX	Numbering Plan Area (more commonly known as area code) NXX (sometimes called exchange) represents the next three numbers of a traditional phone number. The N can be any number from 2-9 and the Xs can be any number. The combination of a phone number's NPA-NXX will usually indicate the physical location of the call device. The exceptions include toll-free numbers and ported number (see LNP).
NTP	Network Time Protocol. An internet standard used for synchronizing clocks of elements distributed on an IP network.
NTSC	National Television Standards Committee. Defines the analog color television broadcast standard used today in North America.
OID	Object Identification.
OSP	Operator Service Provider.
OSS	Operations Systems Support. The back-office software used for configuration, performance, fault, accounting, and security management.
OSS-D	OSS Default. Network Provider Provisioning Server.
PAL	Phase Alternate Line. The European color television format that evolved from the American NTSC standard.
PCES	PacketCable Electronic Surveillance.
PCM	Pulse Code Modulation. A commonly employed algorithm to digitize an analog signal (such as a human voice) into a digital bit stream using simple analog-to-digital conversion techniques.

PDU	Protocol Data Unit.
PHS	Payload Header Suppression. A DOCSIS technique for compressing the Ethernet, IP, and UDP headers of RTP packets.
PKCS	Public-Key Cryptography Standards. Published by RSA Data Security Inc. These Standards describe how to use public key cryptography in a reliable, secure and interoperable way.
PKI	Public-Key Infrastructure. A process for issuing public key certificates, which includes standards, Certification Authorities, communication between authorities and protocols for managing certification processes.
PKINIT	Public-Key Cryptography for Initial Authentication. The extension to the Kerberos protocol that provides a method for using public-key cryptography during initial authentication.
PSC	Payload Service Class Table, a MIB table that maps RTP payload Type to a Service Class Name.
PSFR	Provisioned Service Flow Reference. An SFR that appears in the DOCSIS configuration file.
PSTN	Public Switched Telephone Network.
QCIF	Quarter Common Intermediate Format.
QoS	Quality of Service. Guarantees network bandwidth and availability for applications.
RADIUS	Remote Authentication Dial-In User Service. An internet protocol (IETF RFC 2865 and RFC 2866) originally designed for allowing users dial-in access to the internet through remote servers. Its flexible design has allowed it to be extended well beyond its original intended use.
RAS	Registration, Admissions and Status. RAS Channel is an unreliable channel used to convey the RAS messages and bandwidth changes between two H.323 entities.
RFC	Request for Comments. Technical policy documents approved by the IETF which are available on the World Wide Web at http://www.ietf.cnri.reston.va.us/rfc.html.
RFI	The DOCSIS Radio Frequency Interface specification.
RJ-11	Registered Jack-11. A standard 4-pin modular connector commonly used in the United States for connecting a phone unit into a wall jack.
RKS	Record Keeping Server. The device, which collects and correlates the various Event Messages.
RSA	A public-key, or asymmetric, cryptographic algorithm used to provide authentication and encryption services. RSA stands for the three inventors of the algorithm; Rivest, Shamir, Adleman.
RSA Key Pair	A public/private key pair created for use with the RSA cryptographic algorithm.
RSVP	Resource Reservation Protocol.
RTCP	Real-Time Control Protocol.
RTO	Retransmission Timeout.
RTP	Real-time Transport Protocol. A protocol for encapsulating encoded voice and video streams. Refer to IETF RFC 1889.
SA	Security Association. A one-way relationship between sender and receiver offering security services on the communication flow.
SAID	Security Association Identifier. Uniquely identifies SAs in the DOCSIS Baseline Privacy Plus Interface (BPI+) security protocol.

SCCP	Signaling Connection Control Part. A protocol within the SS7 suite of protocols that provides two functions in addition to those provided within MTP. The first function is the ability to address applications within a signaling point. The second function is Global Title Translation.
SCP	Service Control Point. A Signaling Point within the SS7 network, identifiable by a Destination Point Code that provides database services to the network.
SCTP	Stream Control Transmission Protocol.
SDP	Session Description Protocol.
SDU	Service Data Unit. Information delivered as a unit between peer service access points.
SF	Service Flow. A unidirectional flow of packets on the RF interface of a DOCSIS system.
SFID	Service Flow ID. A 32-bit integer assigned by the CMTS to each DOCSIS Service Flow defined within a DOCSIS RF MAC domain. SFIDs are considered to be in either the upstream direction (USFID) or downstream direction (DSFID). Upstream Service Flow IDs and Downstream Service Flow IDs are allocated from the same SFID number space.
SFR	Service Flow Reference. A 16-bit message element used within the DOCSIS TLV parameters of Configuration Files and Dynamic Service messages to temporarily identify a defined Service Flow. The CMTS assigns a permanent SFID to each SFR of a message.
SG	Signaling Gateway. An SG is a signaling agent that receives/sends SCN native signaling at the edge of the IP network. In particular, the SS7 SG function translates variants ISUP and TCAP in an SS7-Internet Gateway to a common version of ISUP and TCAP.
SGCP	Simple Gateway Control Protocol. Earlier draft of MGCP.
SHA – 1	Secure Hash Algorithm 1. A one-way hash algorithm.
SID	Service ID. A 14-bit number assigned by a CMTS to identify an upstream virtual circuit. Each SID separately requests and is granted the right to use upstream bandwidth.
SIP	Session Initiation Protocol. An application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants.
SIP+	Session Initiation Protocol Plus. An extension to SIP.
S-MTA	Standalone MTA. A single node that contains an MTA and a non-DOCSIS MAC (e.g., ethernet).
SNMP	Simple Network Management Protocol.
SOHO	Small Office/Home Office.
SS7	Signaling System number 7. An architecture and set of protocols for performing out-of-band call signaling with a telephone network.
SSP	Service Switching Point. SSPs are points within the SS7 network that terminate SS7 signaling links and also originate, terminate, or tandem switch calls.
STP	Signal Transfer Point. A node within an SS7 network that routes signaling messages based on their destination address. This is essentially a packet switch for SS7. It may also perform additional routing services such as Global Title Translation.
TCAP	Transaction Capabilities Application Protocol. A protocol within the SS7 stack that is used for performing remote database transactions with a Signaling Control Point.
TCP	Transmission Control Protocol.
TD	Timeout for Disconnect.
TFTP	Trivial File Transfer Protocol.

TFTP-D	Default – Trivial File Transfer Protocol.
TGS	Ticket Granting Server. A sub-system of the KDC used to grant Kerberos tickets.
TGW	Telephony Gateway.
TLS	Transport Layer Security. An IETF proposed standard based on the Secure Socket Layer (SSL) protocol.
TIPHON	Telecommunications and Internet Protocol Harmonization Over Network.
TLV	Type-Length-Value. A tuple within a DOCSIS configuration file.
TN	Telephone Number.
ToD	Time-of-Day Server.
TOS	Type of Service. An 8-bit field of every IP version 4 packet. In a DiffServ domain, the TOS byte is treated as the DiffServ Code Point, or DSCP.
TSG	Trunk Subgroup.
UDP	User Datagram Protocol. A connectionless protocol built upon Internet Protocol (IP).
VAD	Voice Activity Detection.
VBR	Variable Bit Rate.
VoIP	Voice-over-IP.

5 ARCHITECTURAL OVERVIEW OF PACKETCABLE SECURITY

5.1 PacketCable Reference Architecture

Security requirements have been defined for every signaling and media link within the PacketCable IP network. In order to understand the security requirements and specifications for PacketCable, one must first understand the overall architecture. This section presents a brief overview of the PacketCable architecture. For a more detailed overview, refer to the PacketCable 1.5 Architecture Framework Technical Report [1].

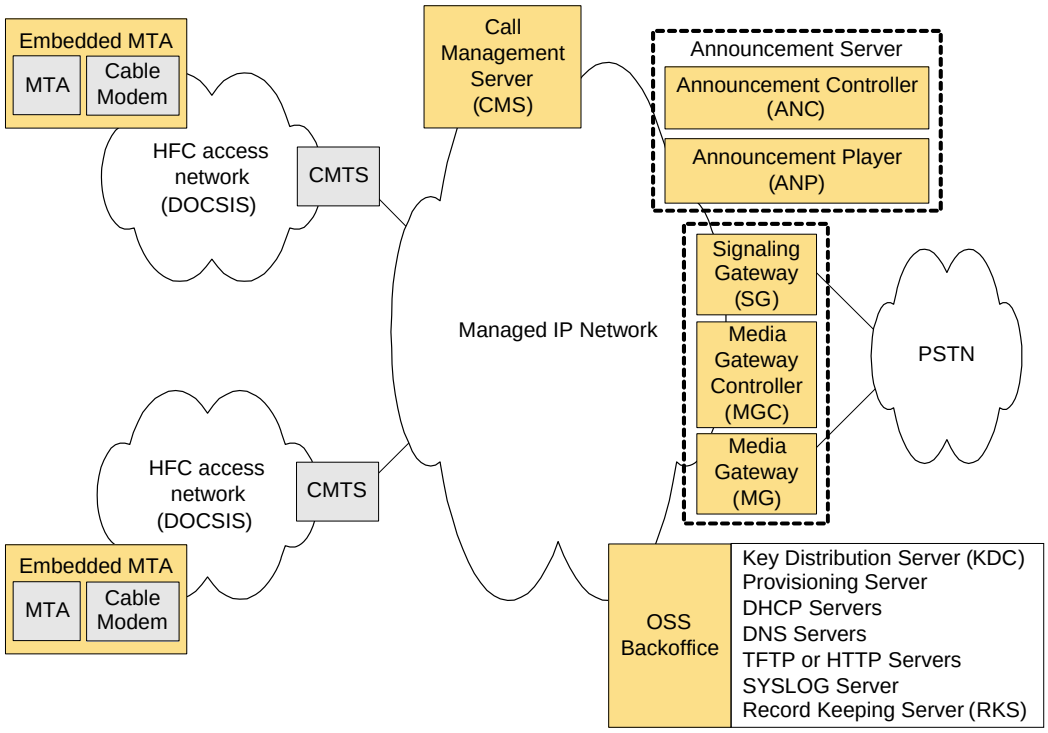


Figure 1. PacketCable Single Zone Architecture

5.1.1 HFC Network

In the above diagram, the Access Network between the MTAs and the CMTS is an HFC network, which employs DOCSIS 1.1 physical layer and MAC layer protocols [8]. DOCSIS BPI+ [9] and QoS protocols are enabled over this link.

5.1.2 Call Management Server

In the context of voice communications applications, a central component of the system is the Call Management Server (CMS). It is involved in both call signaling and the establishment of Dynamic Quality of Service (DQoS). The CMS also performs queries at the PSTN Gateway for LNP (Local Number Portability) and other services necessary for voice communications, including interfacing with the PSTN.

As described in the PacketCable Architecture Framework [1], the CMS is divided into the following functional components:

- Call Agent (CA) – The Call Agent maintains network intelligence and call state and controls the media gateway. Most of the time Call Agent is synonymous for Call Management Server.

- Gate Controller (GC) – The Gate Controller is a logical QoS management component that is typically part of the CMS. The GC coordinates all quality of service authorization and control on behalf of the application service - e.g., voice communications.
- Media Player Controller (MPC) – The MPC initiates and manages all announcement services provided by the Media Player. The MPC accepts requests from the CMS and arranges for the MP to provide the announcement in the appropriate stream so that the user hears the announcement.
- Media Gateway Controller (MGC) – The Media Gateway Controller maintains the gateway's portion of call state for communications traversing the Gateway.

A particular CMS can contain any subset of the above listed functional components.

5.1.3 Functional Categories

The PacketCable Architecture Framework identifies the following functional categories within the architecture:

- MTA device provisioning
- Quality of Service (HFC access network and managed IP backbone)
- Billing interface security
- Security (specified herein)
- Network call signaling (NCS)
- PSTN interconnectivity
- CODEC functionality and media stream mapping
- Audio Server services
- Electronic surveillance (DF interfaces)

In most cases, each functional category corresponds to a particular PacketCable specification document.

5.1.3.1 Device and Service Provisioning

During MTA provisioning, the MTA gets its configuration with the help of the DHCP and TFTP servers, as well as the OSS.

Provisioning interfaces need to be secured and have to configure the MTA with the appropriate security parameters (e.g., customer X.509 certificate signed by the Service Provider). This document specifies the steps in MTA provisioning, but provides detailed specifications only for the security parameters. Refer to [4] for a full specification on MTA provisioning and customer enrollment.

5.1.3.2 Dynamic Quality of Service

PacketCable provides guaranteed Quality of Service (QoS) for each voice communication within a single zone with Dynamic QoS (DQoS) [3].

DQoS is controlled by the Gate Controller function within the CMS and can guarantee Quality of Service within a single administrative domain. The Gate Controller utilizes the COPS protocol to download QoS policy into the CMTS. After that, the QoS reservation is established via DOCSIS 1.1 QoS messaging between the MTA and the CMTS on both sides of the connection.

5.1.3.3 Billing System Interfaces

The CMS, CMTS and the PSTN Gateway are all required to send out billing event messages to the Record Keeping Server (RKS). This interface is currently specified to be RADIUS. Billing information should be

checked for integrity and authenticity as well as kept private. This document defines security requirements and specifications for the communication with RKS.

5.1.3.4 Call Signaling

The call signaling architecture defined within PacketCable is Network Based Call Signaling (NCS). The CMS is used to control call setup, termination and most other call signaling functions. In the NCS architecture [2], the Call Agent function within the CMS is used in call signaling and utilizes the MGCP protocol.

5.1.3.5 PSTN Interconnectivity

The PSTN interface to the voice communications capabilities of the PacketCable network is through the Signaling and Media Gateways (SG and MG). Both of these gateways are controlled with the MGC (Media Gateway Controller). The MGC may be standalone or combined with a CMS. For further detail on PSTN Gateways, refer to [5].

All communications between the MGC and the SG and MG may be over the same-shared IP network and is subject to similar threats (e.g., privacy, masquerade, denial-of-service) that are encountered in other links in the same network. This document defines the security requirements and specifications for the PSTN Gateway links.

When communications from an MTA to a PSTN phone are made, bearer channel traffic is passed directly between an MTA and an MG. The protocols used in this case are RTP and RTCP, as in the MTA-to-MTA case. Both security requirements and specifications are very similar to the MTA-to-MTA bearer requirements and are fully defined in this document. After a voice communication enters the PSTN, the security requirements as well as specifications are based on existing PSTN standards and are out of the scope of this document.

5.1.3.6 CODEC Functionality and Media Stream Mapping

The media stream between two MTAs or between an MTA and a PSTN Gateway utilizes the RTP protocol. Although BPI+ provides privacy over the HFC network, the potential threats within the rest of the voice communications network require that the RTP packets be encrypted end-to-end.[1]

In addition to RTP, there is an accompanying RTCP protocol, primarily used for reporting of RTCP statistics. In addition, RTCP packets may carry CNAME – a unique identifier of the sender of RTP packets. RTCP also defines a BYE message[2] that can be used to terminate an RTP session. These two additional RTCP functions raise privacy and denial-of-service threats. Due to these threats, RTCP security requirements are the same as the requirements for all other end-to-end (SIP+) signaling and are addressed in the same manner.

In addition to MTAs and PSTN Gateways, Media Servers may also participate in the media stream flows. Media Servers are network-based components that operate on media flows to support various voice communications service options. Media servers perform audio bridging, play terminating announcements, provide interactive voice response services, and so on. Both media stream and signaling interfaces to a Media Server are the same as the interfaces to an MTA. For more information on Codec functionality, see [7].

[1] In general, it is possible for an MTA-to-MTA or MTA-to-PSTN connection to cross the networks of several different Service Providers. In the process, this path may cross a PSTN network. This is an exception to the rule, where all RTP packets are encrypted end-to-end. The media traffic inside a PSTN network does not utilize RTP and has its own security requirements. Thus, in this case the encryption would not be end-to-end and would terminate at the PSTN Gateway on both sides of the intermediate PSTN network.

[2] The RTCP BYE message should not be confused with the SIP+ BYE message that is also used to indicate the end of a voice communication within the network.

5.1.3.7 Audio Server Services

Audio Server interfaces provide a suite of signaling protocols for providing announcement and audio services in a PacketCable network.

5.1.3.7.1 Media Player Controller (MPC)

The Media Player Controller (MPC) initiates and manages all announcement services provided by the Media Player. The MPC accepts requests from the CMS and arranges for the MP to provide the announcement in the appropriate stream so that the user hears the announcement. The MPC also serves as the termination for certain calls routed to it for IVR services. When the MP collects information from the end-user, the MPC is responsible for interpreting this information and managing the IVR session accordingly. The MPC manages call state.

5.1.3.7.2 Media Player (MP)

The Media Player (MP) is a media resource server. It is responsible for receiving and interpreting commands from the MPC and for delivering the appropriate announcement(s) to the MTA. The MP provides the media stream with the announcement contents. The MP also is responsible for accepting and reporting user inputs (e.g., DTMF tones). The MP functions under the control of the MPC.

5.1.3.8 Electronic Surveillance

The event interface between the CMS and the DF provides descriptions of calls, necessary to perform wiretapping. This information includes the media stream encryption key and the corresponding encryption algorithm. This event interface uses RADIUS and is similar to the CMS-RKS interface.

The COPS interface between the CMS and the CMTS is used to signal the CMTS to start/stop duplicating media packets to the DF for a particular call. This is the same COPS interface that is used for (DQoS) Gate Authorization messages.

5.2 Threats

Figure 2 below contains the interfaces that were analyzed for security.

There are additional interfaces identified in PacketCable but for which protocols are not specified. In those cases, the corresponding security protocols are also not specified, and those interfaces are not listed in the Figure 2 below.

As well, the interfaces for which security is not required in PacketCable are not listed.

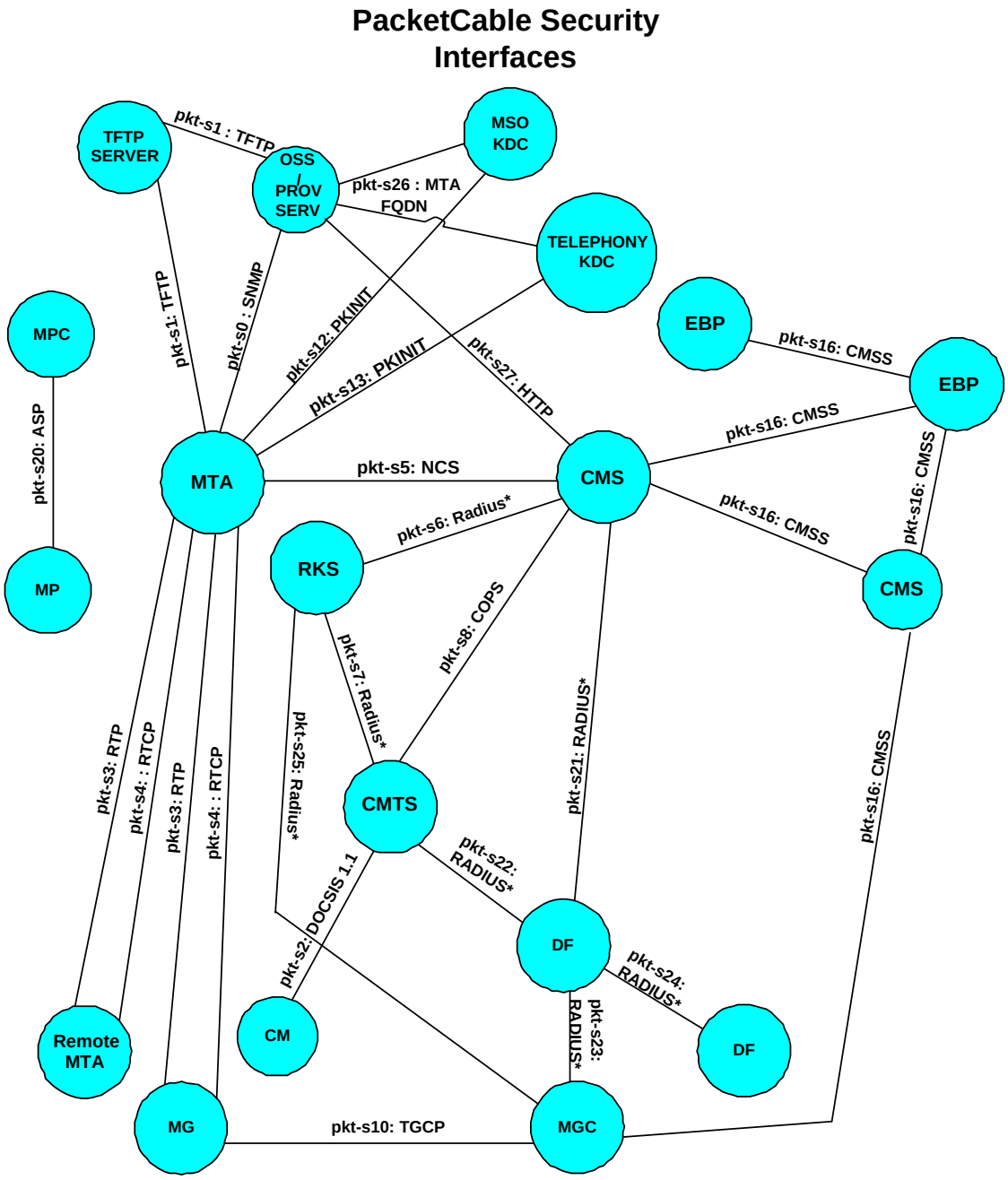


Figure 2. PacketCable Secured Interfaces

NB: The interfaces marked "RADIUS*" carry event messages, which use the RADIUS format as defined by [13].

Following is a summary of general threats and the corresponding attacks that are relevant in the context of IP voice communications. This list of threats is not based on the knowledge of the specific protocols or security mechanisms employed in the network. A more specific summary of threats that are based on the functionality of each network element is listed in Section 5.2.6.

Some of the outlined threats cannot be addressed purely by cryptographic means – physical security and/or fraud management should also be used. These threats may be important, but cannot be fully addressed

within the scope of PacketCable. How vendors and MSOs implement fraud management and physical security will differ and in this case a standard is not required for interoperability.

5.2.1 Theft of Network Services

In the context of voice communications, the main services that may be stolen are:

- Long distance service
- Local (subscription) voice communications service
- Video conferencing
- Network-based three-way calling
- Quality of Service

5.2.1.1 MTA Clones

One or more MTAs can masquerade as another MTA by duplicating its permanent identity and keys. The secret cryptographic keys may be obtained by either breaking the physical security of the MTA or by employing cryptanalysis.

When an MTA is broken into the perpetrator can steal voice communications service and charge it all to the original owner. The feasibility of such an attack depends on where an MTA is located. This attack must be seriously considered in the cases when an MTA is located in an office or apartment building, or on a street corner.

An owner might break into his or her own MTA in at least one instance – after a false account with the MSO providing the voice communications service had been setup. The customer name, address, Social Security Number may all be invalid or belong to someone else. The provided Credit Card Number may be stolen. In that case, the owner of the MTA would not mind giving out the MTA cryptographic identity to others – he or she would not have to pay for service anyway.

In addition to cloning of the permanent cryptographic keys, temporary (usually symmetric) keys may also be cloned. Such an attack is more complex, since the temporary keys expire more often and have to be frequently redistributed. The only reason why someone would attempt this attack is if the permanent cryptographic keys are protected much better than the temporary ones, or if the temporary keys are particularly easy to steal or discover with cryptanalysis.

5.2.1.2 Other Clones

It is conceivable that the cryptographic identity of another network element, such as a CMTS or a CMS, may be cloned. Such an attack is most likely to be mounted by an insider such as a corrupt or disgruntled employee.

5.2.1.3 Subscription Fraud

A customer sets up an account under false information.

5.2.1.4 Non-Payment for Voice Communications Services

A customer stops paying his or her bill, but continues to use the MTA for voice communications service. This can happen if the network does not have an automated method to revoke the customer's access to the network.

5.2.1.5 Protocol Attacks against an MTA

A weakness in the protocol can be manipulated to allow an MTA to authenticate to a network server with a false identity or hijack an existing voice communication. This includes replay and man-in-the-middle attacks.

5.2.1.6 Protocol Attacks against Other Network Elements

A perpetrator might employ similar protocol attacks to masquerade as a different Network Element, such as a CMTS or a CMS. Such an attack may be used in collaboration with cooperating MTAs to steal service.

5.2.1.7 Theft of Services Provided by the MTA

Services such as the support for multiple MTA ports, 3-way calling and call waiting may be implemented entirely in the MTA, without any required interaction with the network.

5.2.1.7.1 Attacks

MTA code to support these services may be downloaded illegally by an MTA clone, in which case the clone has to interact with the network to get the download. In that case, this threat is no different from the network service theft described in the previous section.

Alternatively, downloading an illegal code image using some illegal out-of-band means can also enable these services. Such service theft is much harder to prevent (a secure software environment within the MTA may be required). On the other hand, in order for an adversary to go through this trouble, the price for these MTA-based services has to make the theft worthwhile.

An implication of this threat is that valuable services cannot be implemented entirely inside the MTA without a secure software environment in addition to tamperproof protection for the cryptographic keys. (While a secure software environment within an MTA adds significant complexity, it is an achievable task.)

5.2.1.8 MTA Moved to Another Network

A leased MTA may be reconfigured and registered with another network, contrary to the intent and property rights of the leasing company.

5.2.2 Bearer Channel Information Threats

This class of threats is concerned with the breaking of privacy of voice communications over the IP bearer channel. Threats against non-VoIP communications are not considered here and assumed to require additional security at the application layer.

5.2.2.1 Attacks

Clones of MTAs and other Network Elements, as well as protocol manipulation attacks, also apply in the case of Bearer Channel Information threats. These attacks are already described under the Service Theft threats.

MTA cloning attacks mounted by the actual owner of the MTA are less likely in this case, but not inconceivable. An owner of an MTA may distribute clones to unsuspecting victims, so that he or she can later spy on them.

5.2.2.1.1 Off-line Cryptanalysis

Bearer channel information may be recorded and then analyzed over a period of time, until the encryption keys are discovered through cryptanalysis. The discovered information may be of value even after a relatively long time has passed.

5.2.3 Signaling Channel Information Threats

Signaling information, such as the caller identity and the services to which each customer subscribes may be collected for marketing purposes. The caller identity may also be used illegally to locate a customer that wishes to keep his or her location private.

5.2.3.1 Attacks

Clones of MTAs and other Network Elements, as well as protocol manipulation attacks, also apply in the case of the Signaling Channel Information threats. These attacks were already described under the Service Theft threats.

MTA cloning attacks mounted by the actual owner of the MTA is theoretically possible in this case. An owner of an MTA may distribute clones to the unsuspecting victims, so that he or she can monitor their signaling messages (e.g., for information with marketing value). The potential benefits of such an attack seem unjustified, however.

5.2.3.1.1 Caller ID

A number of a party initiating a voice communication is revealed, even though a number is not generally available (i.e., is "unlisted") and the owner of that number enabled ID blocking.

5.2.3.1.2 Information with Marketing Value

Dialed numbers and the type of service customers use may be gathered for marketing purposes by other corporations.

5.2.4 Service Disruption Threats

This class of threats is aimed at disrupting the normal operation of voice communications. The motives for denial-of-service attacks may be malicious intent against a particular individual or against the service provider. Or, perhaps a competitor wishes to degrade the performance of another service provider and use the resulting problems in an advertising campaign.

5.2.4.1 Attacks

5.2.4.1.1 Remote Interference

A perpetrator is able to manipulate the protocol to close down ongoing voice communications. This might be achieved by masquerading as an MTA involved in such an ongoing communication. The same effect may be achieved if the perpetrator impersonates another Network Element, such as a Gate Controller or an Edge Router during either call setup or voice packet routing.

Depending on the signaling protocol security, it might be possible for the perpetrator to mount this attack from the MTA, in the privacy of his or her own home.

Clones of MTAs and other Network Elements, as well as protocol manipulation attacks, also apply in the case of the Service Disruption threats. These attacks are described under Service Theft threats.

MTA cloning attacks mounted by the actual owner of the MTA can theoretically be used in service disruption against unsuspecting clone owners. However, since there are so many other ways to cause service disruption, such an attack cannot be taken seriously in this context.

5.2.5 Repudiation

In a network where masquerading (using the above-mentioned cloning and protocol manipulation techniques) is common or easily achievable, a customer may repudiate a particular communication (and, thus deny responsibility for paying for it) on that basis.

In addition, unless public key-based digital signatures are employed on each message, the source of each message cannot be absolutely proven. If a signature over a message that originated at an MTA is based on a symmetric key that is shared between that MTA and a network server (e.g., the CMS), it is unclear if the owner of the MTA can claim that the Service Provider somehow falsified the message.

However, even if each message were to carry a public key-based digital signature and if each MTA were to employ stringent physical security, the customer can still claim in court that someone else initiated that communication without his or her knowledge, just as a customer of a telecommunications carrier on the PSTN can claim, e.g., that particular long distance calls made from the customer's telephone were not

authorized by the customer. Such telecommunications carriers commonly address this situation by establishing contractual and/or tariffed relationships with customers in which customers assume liability for unauthorized use of the customer's service. These same contractual principles are typically implemented in service contracts between information services providers such as ISPs and their subscribers. For these reasons, the benefits of non-repudiation seem dubious at best and do not appear to justify the performance penalty of carrying a public key-based digital signature on every message.

5.2.6 Threat Summary

This section provides a summary of the above of threats and attacks and a brief assessment of their relative importance.

5.2.6.1 Primary Threats

Theft of Service. Attacks are:

- **Subscription Fraud**. This attack is prevalent in today's telephony systems (i.e., the PSTN) and requires little economic investment. It can only be addressed with a Fraud Management system.
- **Non-payment for services**. Within the PSTN, telecommunications carriers usually do not prosecute the offenders, but simply shut down their accounts. Because prosecution is expensive and not always successful, it is a poor counter to this attack. Methods such as debit-based billing and device authorization (pay as you play), increasingly common in the wireless sector of the PSTN, might be a possible solution for this attack in the PacketCable context. This threat can also be minimized with effective Fraud Management systems.
- **MTA clones**. This threat requires more technical knowledge than the previous two threats. A technically-knowledgeable adversary or underground organization might offer cloning services for profit. This threat is most effective when combined with subscription fraud, where an MTA registered under a fraudulent account is cloned. This threat can be addressed with both Fraud Management and physical security inside the MTA, or a combination of both.
- **Impersonate a network server.** With proper cryptographic mechanisms, authorization and procedural security in place, this attack is unlikely, but has the potential for great damage.
- **Protocol manipulation.** Can occur only when security protocols are flawed or when not enough cryptographic strength is in place.

Bearer Channel Information Disclosure. Attacks are:

- **Simple Snooping. This would happen if voice packets were sent in the clear over some segment of the network. Even if that segment appears to be protected, an insider may still compromise it. This is the only major attack on privacy. The bearer channel privacy attacks listed below are possible but are all of secondary importance.**
- **MTA clones. Again, this threat requires more technical knowledge but can be offered as a service by an underground organization. A most likely variation of this attack is when a publicly accessible MTA (e.g., in an office or apartment building) is cloned.**
- **Protocol manipulation. A flawed protocol may somehow be exploited to discover bearer channel encryption keys.**
- **Off-line cryptanalysis. Even when media packets are protected with encryption, they can be stored and analyzed for long periods of time, until the decryption key is finally discovered. Such an attack is not likely to be prevalent, since it is justified only for particularly valuable customer-provided information (PacketCable security is not required to protect data). This attack is more difficult to perform on voice packets (as opposed to data). Still, customers are very sensitive to this threat and it can serve as the basis for a negative publicity campaign by competitors.**

Signaling Information Disclosure. This threat is listed as primary only due to potential for bad publicity and customer sensitivity to keeping their numbers and location private. All of the attacks listed below are similar to those for bearer channel privacy and are not described here:

- **Simple snooping**
- **MTA clones**
- **Protocol manipulation**
- **Off-line cryptanalysis**
- **Service disruption**

5.2.6.2 Secondary Threats.

- **Theft of MTA-based services.** Based on the voice communications services that are planned for the near future, this threat does not appear to have potential for significant economic damage. This could possibly change with the introduction of new value-added services in the future.
- **Illegally registering a leased MTA with a different Service Provider.** Leased MTAs can normally be tracked. Most likely, this threat is combined with the actual theft of a leased MTA. Thus, this threat does not appear to have potential for widespread damage.

5.3 Security Architecture

5.3.1 Overview of Security Interfaces

The diagram below summarizes all of the PacketCable security interfaces, including key management.

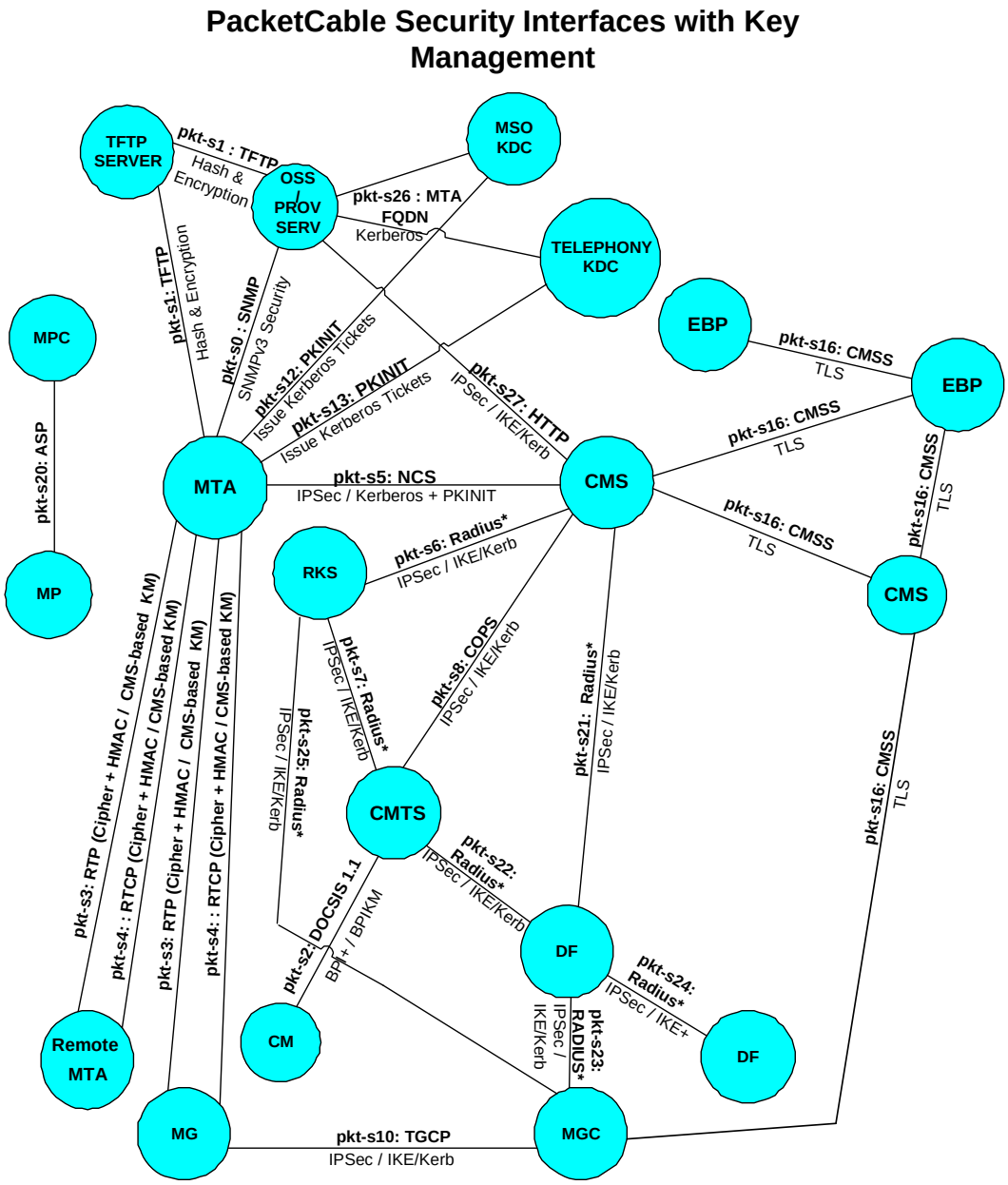


Figure 3. PacketCable Security Interfaces with Key-Management

In the above diagram, each interface label is of the form:

<**label**>: **<protocol> { <security protocol> / <key management protocol> }**

If the key management protocol is missing, it is not needed for that interface. PacketCable interfaces that do not require security are not shown on this diagram.

The following abbreviations are used in the above diagram:

IKE/Kerb	IKE (with pre-shared keys or X.509 certificates) or Kerberos
IKE+	IKE with X.509 certificates

CMS-based KM	Keys randomly generated and exchanged inside signaling messages
RADIUS*	Event messages, which use the RADIUS format as defined by [13].

The following table briefly describes each of the interfaces shown in the above diagram:

Table 1. PacketCable Security Interfaces Table

Interface	Components	Description
pkt-s0	MTA – PS/OSS	Immediately after the DHCP sequence in the Secure Provisioning Flow, the MTA performs Kerberos-based key management with the Provisioning Server to establish SNMPv3 keys. The MTA bypasses Kerberized SNMPv3 and uses SNMPv2c in the Basic and Hybrid Flows.
pkt-s1	MTA – TFTP	MTA Configuration file download. When the Provisioning Server in the Secure Provisioning Flow sends an SNMP Set command to the MTA, it includes both the configuration name and the hash of the file. Later, when the MTA downloads the file, it authenticates the configuration file using the hash value. The configuration file may be optionally encrypted.
pkt-s2	CM – CMTS	DOCSIS 1.1: This interface should be secured with BPI+ using BPI Key Management. BPI+ privacy is provided on the HFC link.
pkt-s3	MTA – MTA MTA – MG	RTP: End-to-end media packets between two MTAs, or between MTA and MG. RTP packets are encrypted directly with the chosen cipher. Message integrity is optionally provided by an MMH MAC. Keys are randomly generated, and exchanged by the two endpoints inside the signaling messages via the CMS or other application server.
pkt-s4	MTA – MTA MTA – MG	RTCP: RTCP control protocol for RTP. Message integrity and encryption by selected cipher. The RTCP keys are derived using the same secret negotiated during the RTP key management. No additional key management messages are needed or utilized.
pkt-s5	MTA – CMS	NCS: Message integrity and privacy via IPsec. Key management is with Kerberos with PKINIT (public key initial authentication) extension.
pkt–s6	RKS – CMS	RADIUS:IPsec is used for both message integrity and privacy. Key management is IKE or Kerberos.
pkt-s7	RKS – CMTS	RADIUS: IPsec is used for both message integrity and privacy. Key management is IKE or Kerberos.
pkt-s8	CMS – CMTS	COPS: COPS protocol between the GC and the CMTS, used to download QoS authorization to the CMTS. IPsec is used for message integrity, as well as privacy. Key management is IKE or Kerberos.
pkt-s9		This interface has been removed from the PacketCable architecture.
pkt-s10	MGC – MG	TGCP: PacketCable interface to the PSTN Media Gateway. IPsec is used for both message integrity and privacy. Key management is IKE or Kerberos.
pkt-s11		This interface has been removed from the PacketCable architecture.

Interface	Components	Description
pkt-s12	MTA – MSO KDC	PKINIT: An AS-REQ message is sent to the KDC with public-key cryptography used for authentication. The KDC verifies the certificate and issues either a service ticket or a ticket granting ticket (TGT), depending on the contents of the AS Request. The AS Reply returned by the KDC contains a certificate chain and a digital signature that are used by the MTA to authenticate this message. In the case that the KDC returns a TGT, the MTA then sends a TGS Request to the KDC to which the KDC replies with a TGS Reply containing a service ticket. The TGS Request/Reply messages are authenticated using a symmetric session key inside the TGT.
pkt-s13	MTA – Telephony KDC	PKINIT: See pkt-s12 above.
pkt-s14		This interface has been removed from the PacketCable architecture.
pkt-s15		This interface has been removed from the PacketCable architecture.
pkt-s16	CMS – CMS CMS – MGC CMS – EBP EBP – EBP	SIP: TLS is used for both message integrity and privacy. Certificates are used for mutual authentication during the TLS handshake.
pkt-s17		This interface has been removed from the PacketCable architecture.
pkt-s18		This interface has been removed from the PacketCable architecture.
pkt-s19		This interface has been removed from the PacketCable architecture.
pkt-s20	MPC – MP	ASP: IPsec is used for both message integrity and privacy. Key management is IKE or Kerberos.
pkt-s21	DF – CMS	RADIUS: IPsec is used for both message integrity and privacy. Key management is IKE or Kerberos.
pkt-s22	DF – CMTS	RADIUS: IPsec is used for both message integrity and privacy. Key management is IKE or Kerberos.
pkt-s23	DF – MGC	RADIUS: IPsec is used for both message integrity and privacy. Key management is IKE or Kerberos.
pkt-s24	DF – DF	RADIUS: IPsec is used for both message integrity and privacy. Key management is IKE+.
pkt-s25	RKS – MGC	RADIUS: IPsec is used for both message integrity and privacy. Key management is IKE or Kerberos.
pkt-s26	OSS/Prov Serv – MSO KDC OSS/Prov Serv – Telephony KDC	The KDC uses Kerberos to map the MTA's MAC address to its FQDN for the purpose of authenticating the MTA before issuing it a ticket.
pkt-s27	CMS-PS/OSS	HTTP: IPsec is used for both message integrity and privacy. Key management is IKE or Kerberos.
pkt-s28		This interface has been removed from the PacketCable architecture.

5.3.2 Security Assumptions

5.3.2.1 BPI+ CMTS Downstream Messages Are Trusted

As mentioned previously, it is assumed that CMTS downstream messages cannot be easily modified in transit and a CMTS can be impersonated only at great expense.

Most messages secured in this specification either move over the shared IP network in addition to the DOCSIS path, or do not go over DOCSIS at all.

In one case – the case of DOCSIS QoS messages exchanged between the CMTS and the CM – this assumption does not apply. Although DOCSIS QoS messages (both upstream and downstream) include an integrity check, the corresponding (BPI+) key management does not authenticate the identity of the CMTS. The CM is unable to cryptographically know that the network element it has connected to is the true CMTS for that network. However, even if a CMTS could be impersonated, it would allow only limited denial-of-service attacks. This vulnerability is not considered to be worth the effort and the expense of impersonating a CMTS.

5.3.2.2 Non-Repudiation Not Supported

Non-repudiation, in this specification, means that an originator of a message cannot deny that he or she sent that message. In this voice communications architecture, non-repudiation is not supported for most messages, with the exception of the top key management layer. This decision was based on the performance penalty incurred with each public key operation. The most important use for non-repudiation would have been during communications setup – to prove that a particular party had initiated that particular communication. However, due to very strict requirements on the setup time, it is not possible to perform public key operations for each communication.

5.3.2.3 Root Private Key Compromise Protection

The cryptographic mechanisms defined in this document are based on a Public Key Infrastructure (PKI). As is the case with most other architectures that are based on a PKI, there is no automated recovery path from a compromise of a Root Private Key. However, with proper safeguards, the probability of this happening is very low, to the point that the risk of a root private key compromise occurring is outweighed by the benefits of this architecture.

The corresponding Root Public Key is stored as a read-only parameter in many components of this architecture. Once the Root Private Key has been compromised, each manufacturer's certificate would have to be manually reconfigured.

Due to this limitation of a PKI, the Root Private Key must be very carefully guarded with procedural and physical security. And, it must be sufficiently long so that its value cannot be discovered with cryptographic attacks within the expected lifetime of the system.

5.3.2.4 Limited Prevention of Denial-of-Service Attacks

This document does not attempt to address all or even most denial-of-service attacks. The cryptographic mechanisms defined in this specification prevent some denial-of-service attacks that are particularly easy to mount and are hard to detect. For example, they will prevent a compromised MTA from masquerading as other MTAs in the same upstream HFC segment and interrupting ongoing communications with illicit HANGUP messages.

This specification will also prevent more serious denial-of-service attacks, such as an MTA masquerading as a CMS in a different network domain that causes all communications setup requests to fail.

On the other hand, denial-of-service attacks where a router is taken out of service or is bombarded with bad IP packets are not addressed. In general, denial-of-service attacks that are based on damaging one of the network components can only be solved with procedural and physical security, which is out of the scope of this specification.

Denial-of-service attacks where network traffic is overburdened with bad packets cannot be prevented in a large network (although procedural and physical security helps), but can usually be detected. Detection of such an attack and of its cause is out of scope of this specification.

For example, denial-of-service attacks where a router is taken out of order or is bombarded with bogus IP packets cannot be prevented.

5.3.3 Susceptibility of Network Elements to Attack

This section describes the amount and the type of trust that can be assumed for each element of the voice communications network. It also describes the specific threats that are possible if each network component is compromised. These threats are based on the functionality specified for each component. The general categories of threats are described in Section 5.2.

Both the trust and the specific threats are described with the assumption that no cryptographic or physical security has been employed in the system, with the exception of the BPI+ security that is assumed on the HFC DOCSIS links. The goal of this security specification is to address threats that are relevant to this voice communications system.

5.3.3.1 Managed IP Network

It is assumed that the same IP network may be shared between multiple, possibly competing service providers. It is also assumed that the service provider may provide multiple services on the same IP network, e.g., Internet connectivity. No assumptions can be made about the physical security of each link in this IP network. An intruder can pop up at any location with the ability to monitor traffic, perform message modification and to reroute messages.

5.3.3.2 MTA

The MTA is considered to be an untrusted network element. It is operating inside customer premises, considered to be a hostile environment. It is assumed that a hostile adversary has the ability to open up the MTA and make software and even hardware modifications to fit his or her needs. This would be done in the privacy of the customer's home.

The MTA communicates with the CMTS over the shared DOCSIS path and has access to downstream and upstream messages from other MTAs within the same HFC segment.

An MTA is responsible for:

- Initiating and receiving communications to/from another MTA or the PSTN
- Negotiating QoS

A compromise of an MTA can result in:

- MTA clones that are capable of:
  - Accessing basic service and any enhanced features in the name of another user's account
  - Violating privacy of the owner of the compromised MTA that doesn't know that the keys were stolen
  - Identity fraud
- An MTA running a bad code image that disrupts communications made by other MTAs or degrades network performance

5.3.3.3 CMTS

The CMTS communicates both over the DOCSIS path and over the shared IP network. When the CMTS sends downstream messages over the DOCSIS path, it is assumed that a perpetrator cannot modify them or impersonate the CMTS. BPI+ over that path provides privacy.

However, when the CMTS is communicating over the shared IP network (e.g., with the CMS or another CMTS), no such assumptions can be made.

While the CMTS, as well as voice communications network servers are more trusted than the MTAs, they cannot be trusted completely. There is always a possibility of an insider attack.

Insider attacks at the CMTS should be addressed by cryptographic authentication and authorization of the CMTS operators, as well as by physical and procedural security, which are all out of the scope of the PacketCable specifications.

A CMTS is responsible for:

- Reporting billing-related statistics to the RKS
- QoS allocation for MTAs over the DOCSIS path
- Implementation of BPI+ (MAC layer security) and corresponding key management

A compromise of a CMTS may result in:

- Service theft by reporting invalid information to the RKS
- Unauthorized levels of QoS
- Loss of privacy, since the CMTS holds BPI+ keys. This may not happen if additional encryption is provided above the MAC layer
- Degraded performance of some or all MTAs in that HFC segment
- Some or all of the MTAs in one HFC segment completely taken out of service

5.3.3.4 Voice Communications Network Servers are Untrusted Network Elements

Application servers used for voice communications (e.g., CMS, RKS, Provisioning, OSS, DHCP and TFTP Servers) reside on the network and can potentially be impersonated or subjected to insider attacks. The main difference would be in the damage that can be incurred in the case a particular server is impersonated or compromised.

Threats that are associated with each network element are discussed in the following subsections. To summarize those threats, a compromise or impersonation of each of these servers can result in a wide-scale service theft, loss of privacy, and in highly damaging denial-of-service attacks.

In addition to authentication of all messages to and from these servers (specified in this document), care should be taken to minimize the likelihood of insider attacks. They should be addressed by cryptographic authentication and authorization of the operators, as well as by stringent physical and procedural security, which are all out of scope of the PacketCable specifications.

5.3.3.4.1 CMS

The Call Management Server is responsible for:

- Authorizing individual voice communications by subscribers
- QoS allocation
- Initializing the billing information in the CMTS
- Distributing per communication keys for MTA-MTA signaling, bearer channel, and DQoS messages on the MTA-CMTS and CMTS-CMTS links
- Interface to PTSN gateway

A compromised CMS can result in:

- Free voice communications service to all of the MTAs that are located in the same network domain (up to 100,000). This may be accomplished by:
    - Allowing unauthorized MTAs to create communications
    - Uploading invalid or wrong billing information to the CMTS
    - Combination of both of the above

- Loss of privacy, since the CMS distributes bearer channel keys
- Unauthorized allocation of QoS
- Unauthorized disclosure of customer identity, location (e.g., IP address), communication patterns, and a list of services to which the customer subscribes

5.3.3.4.2 RKS

The RKS is responsible for collecting billing events and reporting them to the billing system. A compromised RKS may result in:

- Free or reduced-rate service due to improper reporting of statistics
- Billing to a wrong account
- Billing customers for communications that were never made, i.e., fabricating communications
- Unauthorized disclosure of customer identity, personal information, service usage patterns, and a list of services to which the customer subscribes

5.3.3.4.3 OSS, DHCP & TFTP Servers

The OSS system is responsible for:

- MTA and service provisioning
- MTA code downloads and upgrades
- Handling service change requests and dynamic reconfiguration of MTAs

A compromise of the OSS, DHCP or TFTP server can result in:

- MTAs running illegal code, which may:
    - Intentionally introduce bugs or render the MTA completely inoperable
    - Degrade voice communications performance on the PacketCable or HFC network
    - Configure the MTA with features to which the customer is not entitled
- MTAs configured with an identity and keys of another customer
- MTAs configured with service options for which the customer did not pay
- MTAs provisioned with a bad set of parameters that would make them perform badly or not perform at all

5.3.3.5 PSTN Gateways

5.3.3.5.1 Media Gateway

The MG is responsible for:

- Passing media packets between the PacketCable network and the PSTN
- Reporting statistics to the RKS

A compromise of the MG may result in:

- Service theft by reporting invalid information to the RKS
- Loss of privacy on communications to/from the PSTN

5.3.3.5.2 Signaling Gateway

The SG is responsible for translating call signaling between the PacketCable network and the PSTN.

A compromise of the SG may result in:

- Incorrect MTA identity reported to the PSTN
- Unauthorized services enabled within the PSTN
- Loss of PSTN connectivity
- Unauthorized disclosure of customer identity, location (e.g., IP address), usage patterns and a list of services to which the customer subscribes

6 SECURITY MECHANISMS

Unless explicitly stated otherwise, the following requirements apply to messages described by this document:

1. ASN.1 encoded messages and objects MUST conform to the Distinguished Encoding Rules [36].
2. FQDNs used as components of principal names and principal identifiers MUST be rendered in lower case.
3. FQDNs MUST NOT include the root domain (i.e., they MUST NOT include a trailing dot).
4. All Kerberos messages in PacketCable MUST utilize only UDP/IP.

6.1 IPsec

6.1.1 Overview

IPsec provides network-layer security that runs immediately above the IP layer in the protocol stack. It provides security for the TCP or UDP layer and above. It consists of two protocols, IPsec ESP and IPsec AH, as specified in [19].

IPsec ESP provides confidentiality and message integrity, IP header not included. IPsec AH provides only message integrity, but that includes most of the IP header (with the exception of some IP header parameters that can change with each hop). PacketCable utilizes only the IPsec ESP protocol [20], since authentication of the IP header does not significantly improve security within the PacketCable architecture.

Each protocol supports two modes of use: transport mode and tunnel mode. PacketCable only utilizes IPsec ESP transport mode. For more detail on IPsec and these two modes, refer to [19]. Note that in [19], all implementations of ESP are required to support the concept of Security Associations (SAs). [19] also provides a general model for processing IP traffic relative to SAs. Although particular IPsec implementations need not follow the details of this general model, the external behavior of any IPsec implementation must match the external behavior of the general model. This ensures that components do not accept traffic from unknown addresses and do not send or accept traffic without security (when security is required). PacketCable components that implement IPsec are expected to provide behavior that matches the general model described in [19].

6.1.2 PacketCable Profile for IPsec ESP (Transport Mode)

6.1.2.1 IPsec ESP Transform Identifiers

IPsec Transform Identifier (1 byte) is used by IKE to negotiate an encryption algorithm that is used by IPsec. A list of available IPsec Transform Identifiers is specified in [21]. Within PacketCable, the same Transform Identifiers are used by all IPsec key management protocols: IKE, Kerberos and application layer (embedded in IP signaling messages).

The following table describes the IPsec Transform Identifiers (all of which use the CBC mode specified in [22]) supported by PacketCable.

Table 2. IPsec ESP Transform Identifiers

Transform ID	Value (Hex)	Key Size (in bits)	MUST Support	Description
ESP_3DES	0x03	192	yes	3-DES in CBC mode.
ESP_RC5	0x04	128	no	RC5 in CBC mode
ESP_IDEA	0x05	128	no	IDEA in CBC mode
ESP_CAST	0x06	128	no	CAST in CBC mode
ESP_BLOWFISH	0x07	128	no	BLOWFISH in CBC mode
ESP_NULL	0x0B	0	yes	Encryption turned off
ESP_AES	0x0C	128	no	AES-128 in CBC mode with 128-bit block size

The ESP_3DES and ESP_NULL Transform IDs MUST be supported. ESP_AES is included as an optional encryption algorithm. For all of the above transforms, the CBC Initialization Vector (IV) is carried in the clear inside each ESP packet payload [22]. AES-128 [35] MUST be used in CBC mode with a 128-bit block size and a randomly generated Initialization Vector (IV). AES-128 requires 10 rounds of cryptographic operations [35].

IKE allows negotiation of the encryption key size. Other IPsec Key Management protocols used by PacketCable do not allow key size negotiation, and so for consistency a single key size is listed for each Transform ID. If in the future it is desired to increase the key size for one of the above algorithms, IKE will use the built-in key-size negotiation, while other key management protocols will utilize a new Transform ID for the larger key size.

6.1.2.2 IPsec ESP Authentication Algorithms

The IPsec Authentication Algorithm (1-byte) is used by IKE to negotiate a packet-authentication algorithm that is used by IPsec. A list of available IPsec Authentication Algorithms is specified in [21]. Within PacketCable, the same Authentication Algorithms are used by all IPsec key management protocols: IKE, Kerberos and application layer (embedded in IP signaling messages).

PacketCable supports the following IPsec Authentication Algorithms:

Table 3. IPsec Authentication Algorithms

Authentication Algorithm	Value (Hex)	Key Size (in bits)	MUST Support	Description
HMAC-MD5-96	0x01	128	yes (also required by [21])	First 12 bytes of the HMAC-MD5 as described in [37]
HMAC-SHA-1-96	0x02	160	yes	First 12 bytes of the HMAC-SHA1 as described in [23]

The HMAC-MD5-96 and HMAC-SHA-1-96 authentication algorithms MUST be supported.

6.1.2.3 Replay Protection

In general, IPsec provides an optional replay-protection service (anti-replay service). An IPsec sequence number outside of the current anti-replay window is flagged as a replay and the packet is rejected. When the anti-replay service is turned on, an IPsec sequence number cannot overflow and roll over to 0. Before that happens, a new Security Association must be created as specified in [20].

Within PacketCable Security Specification, the IPsec anti-replay service MUST be turned on at all times. This is regardless of which key-management mechanism is used with the particular IPsec interface.

6.1.2.4 Key Management Requirements

Within PacketCable, IPsec is used on a number of different interfaces with different security and performance requirements. Because of this, several different key management protocols have been chosen for different PacketCable interfaces. On some interfaces it is IKE (see Section 6.2), on other interfaces it is Kerberos/PKINIT (see Section 6.4).

When IKE is not used for key management, an alternative key management protocol needs an interface to the IPsec layer in order to create/update/delete IPsec Security Associations (SAs). IPsec Security Associations MUST be automatically established or re-established as required. This implies that the IPsec layer also needs a way to signal a key management application when a new Security Association needs to be set up (e.g., the old SA is about to expire or there is no SA on a particular interface).

In addition, some network elements are required to run multiple key management protocols. In particular, the Application Server (such as a CMS) and the MTA must support multiple key management protocols. The MTA MUST support Kerberos/PKINIT on the MTA-CMS signaling interface. IKE MUST be supported on the CMS-CMTS and CMS-RKS interfaces.

The PF_KEY interface (see [29]) SHOULD be used for IPsec key management within PacketCable and would satisfy the above listed requirements. For example, PF_KEY permits multiple key management applications to register for rekeying events. When the IPsec layer detects a missing Security Association, it signals the event to all registered key-management applications. Based on the Identity Extension associated with that Security Association, each key-management application decides if it should handle the event.

6.2 Internet Key Exchange (IKE)

6.2.1 Overview

PacketCable utilizes IKE as one of the key management protocols for IPsec [24]. It is utilized on interfaces where:

- There is not a very large number of connections
- The endpoints on each connection know about each other’s identity in advance

Within PacketCable, IKE key management is completely asynchronous to call signaling messages and does not contribute to any delays during communications setup. The only exception would be some unexpected error, where Security Association is unexpectedly lost by one of the endpoints.

IKE is a peer-to-peer key management protocol. It consists of 2 phases. In the first phase, a shared secret is negotiated via a Diffie-Hellman key exchange. It is then used to authenticate the second IKE phase. The second phase negotiates another secret, used to derive keys for the IPsec ESP protocol.

6.2.2 PacketCable Profile for IKE

6.2.2.1 First IKE Phase

There are several modes defined for authentication during the first IKE phase.

6.2.2.1.1 IKE Authentication with Signatures

In this mode, both peers MUST be authenticated with X.509 certificates and digital signatures. PacketCable utilizes this IKE authentication mode on some IPsec interfaces. Whenever this mode is utilized, both sides MUST exchange X.509 certificates (although this is optional in [24]).

6.2.2.1.2 IKE Authentication with Public-Key Encryption

PacketCable MUST NOT utilize this IKE authentication with public key encryption. In order to perform this mode of IKE authentication, the initiator must already have the responder's public key, which is not supported by PacketCable.

6.2.2.1.3 IKE Authentication with Pre-Shared Keys

A key derived by some out-of-band (e.g., manual) mechanism is used to authenticate the exchange. PacketCable utilizes this IKE authentication mode on some IPsec interfaces. PacketCable does not specify the out-of-band method for deriving pre-shared keys.

When using pre-shared keys, the strength of the system is dependent upon the strength of the shared secret. The goal is to keep the shared secret from being the weak link in the chain of security. This implies that the shared secret needs to contain as much entropy (randomness) as the cipher being used. In other words, the shared secret should have at least 128-160 bits of entropy. This means if the shared secret is just a string of random 8-bit bytes, then of the key can be 16-20 bytes. If the shared secret is derived from a passphrase that is a string of random alpha-numerics (a-zA-Z0-9/+), then it should be at least 22-27 characters. This is because there are only 64 characters (6 bits) instead of 256 characters (8 bits) per 8-bit byte, which implies an expansion of 4/3 the length for the same amount of entropy. Both random 8-bit bytes and random 6-bit bytes assume truly random numbers. If there is any structure in the password/passphrase, like deriving from English, then even longer passphrases are necessary. A passphrase composed of English would need on the order of 60-100 characters, depending on mixing of case. Using English passphrases (or any language, for that matter) creates the problem that, if an attacker knows the language of the passphrase then they have less space to search. It is less random. This implies fewer bits of entropy per character, so a longer passphrase is required to maintain the same level of entropy.

6.2.2.2 Second IKE Phase

In the second IKE phase, an IPsec ESP SA is established, including the IPsec ESP keys and ciphersuites. It is possible to establish multiple Security Associations with a single second-phase IKE exchange.

First, a shared second phase secret is established, and then all the IPsec keying material is derived from it using the one-way function specified in [24].

The second-phase secret is built from encrypted nonces that are exchanged by the two parties. Another Diffie-Hellman exchange may be used in addition to the encrypted nonces. Within PacketCable, IKE MUST NOT perform a Diffie-Hellman exchange in the second IKE phase in order to avoid the associated performance penalties.

The second IKE phase is authenticated using a shared secret that was established in the first phase. Supported authentication algorithms are the same as those specified for IPsec in Section 6.1.2.2.

6.2.2.3 Encryption Algorithms for IKE Exchanges

Both phase 1 and phase 2 IKE exchanges include some symmetrically-encrypted messages. The encryption algorithms supported as part of the PacketCable Profile for IKE MUST be the same algorithms identified in the PacketCable profile for IPsec ESP in Table 2 of Section 6.1.2.1.

6.2.2.4 Diffie-Hellman Groups

IKE defines specific sets of Diffie-Hellman parameters (i.e., prime and generator) that may be used for the phase 1 IKE exchanges. These are called groups in [24]. The use of Diffie-Hellman groups within PacketCable IKE is identical to that specified in [24]: the first group MUST be supported and the remaining groups SHOULD be supported. Note that this is different from the requirements pertaining to the PacketCable use of groups in PKINIT described in Section 6.4.2.1.1. Appendix G provides details of the first and second Oakley groups.

6.2.2.5 Security Association Renegotiation

Renegotiation or rekeying of an IKE Security Association (SA) is triggered by the end of its lifetime as measured in elapsed time or number of kilobytes of data protected by the SA. Each peer should transition from the old SA to the new SA the same way to avoid interoperability problems. After successful IKE phase 1 ISAKMP SA renegotiation both peers MUST use the new SA when sending traffic and be able to receive traffic on the new SA. After successful IKE phase 2 IPsec SA renegotiation both peers MUST use the new outbound SA when sending traffic and be able to receive traffic on the new inbound SA.

6.3 SNMPv3

Any mention of SNMP in this specification without a specific reference to the SNMP protocol version must be interpreted as SNMPv3.

PacketCable supports use of SNMPv2c coexistence for network management operations for devices provisioned under the Basic Flow or the Hybrid Flow. It also supports the SNMPv3/v2c coexistence for network management operations when the device is provisioned under the Secure Flow. Refer to the provisioning specification [4] for the use of SNMP coexistence in PacketCable.

For any interface within the PacketCable architecture utilizing SNMPv3, SNMPv3 authentication MUST be turned on at all times and SNMPv3 privacy MAY also be utilized.

In order to establish SNMPv3 keys, all PacketCable SNMP interfaces SHOULD utilize Kerberized SNMPv3 key management (as specified in Section 6.5.4). In addition, SNMPv3 key management techniques specified in [28] MAY also be used.

6.3.1 SNMPv3 Transform Identifiers

The SNMPv3 Transform Identifier (1 byte) is used by Kerberized key management to negotiate an encryption algorithm for use by SNMPv3.

For PacketCable, the following SNMPv3 Transform Identifiers are supported:

Table 4. SNMPv3 Transform Identifiers

Transform ID	Value (Hex)	Key Size (in bits)	MUST be Supported	Description
SNMPv3_DES	0x21	128	yes	DES in CBC mode. The first 64 bits are used as the DES Key and the remaining 64 are used as the pre-IV as described in [28].
SNMPv3_NULL	0x20	0	yes	Encryption turned off

The SNMPv3_DES and the SNMPv3_NULL Transform IDs MUST be supported. The DES encryption transform for SNMPv3 is specified in [28]. Note that DES encryption does not provide strong privacy but is currently the only encryption algorithm specified by the SNMPv3 standard.

6.3.2 SNMPv3 Authentication Algorithms

SNMPv3 Authentication Algorithm (1 byte) is used by Kerberized key management to negotiate an SNMPv3 message authentication algorithm.

For PacketCable, the following SNMPv3 Authentication Algorithms are supported (both of which are specified in [28]):

Table 5. SNMPv3 Authentication Algorithms

Authentication Algorithm	Value (Hex)	Key Size (in bits)	MUST be supported	Description
SNMPv3_HMAC-MD5	0x21	128	yes (also required by [28])	MD5 HMAC
SNMPv3_HMAC-SHA-1	0x22	160	no (SHOULD be supported)	SHA-1 HMAC

The SNMPv3_HMAC-MD5 Authentication Algorithm MUST be supported. The SNMPv3_HMAC-SHA-1 Authentication Algorithm SHOULD be supported.

6.4 Kerberos / PKINIT

6.4.1 Overview

PacketCable utilizes the concept of Kerberized IPsec for signaling between an Application Server, such as the CMS, and the MTA. This refers to the ability to create IPsec Security Associations using keys derived from the subkeys exchanged using the Kerberos AP Request/AP Reply messages. On this interface, Kerberos (Appendix B) is utilized with the PKINIT public key extension (also see Appendix C).

Kerberized IPsec consists of three distinct phases:

1. A client SHOULD obtain a TGT (Ticket Granting Ticket) from the KDC (Key Distribution Center). Once the client obtains the TGT, it MUST use the TGT in the subsequent phase to authenticate to the KDC and obtain a ticket for the specific Application Server, e.g., a CMS.

   In Kerberos, tickets are symmetric authentication tokens encrypted with a particular server's key. (For a TGT, the server is the KDC.) Tickets are used to authenticate a client to a server. A PKI equivalent of a ticket would be an X.509 certificate. In addition to authentication, a ticket is used to establish a session key between a client and a server, where the session key is contained in the ticket.

   The logical function within the KDC that is responsible for issuing TGTs is referred to as an Authentication Server or AS.

2. A client obtains a ticket from the KDC for a specific Application Server. In this phase, a client can authenticate with a TGT obtained in the previous phase. A client can also authenticate to the KDC directly using a digital certificate or a password-derived key, bypassing phase 1.

   The logical function within the KDC that is responsible for issuing Application Server tickets based on a TGT is referred to as the Ticket Granting Server – TGS. When the TGT is bypassed, it is the Authentication Server that issues the Application Server tickets.

3. A client utilizes the ticket obtained in the previous phase to establish a pair of Security Parameters (one to send and one to receive) with the server. This is the only key management phase that is not already specified in an IETF standard. The previous two phases are part of standard Kerberos, while this phase defines new messages that tie together Kerberos key management and IPsec.

The following diagram illustrates the three phases of Kerberos-based key management for IPsec:

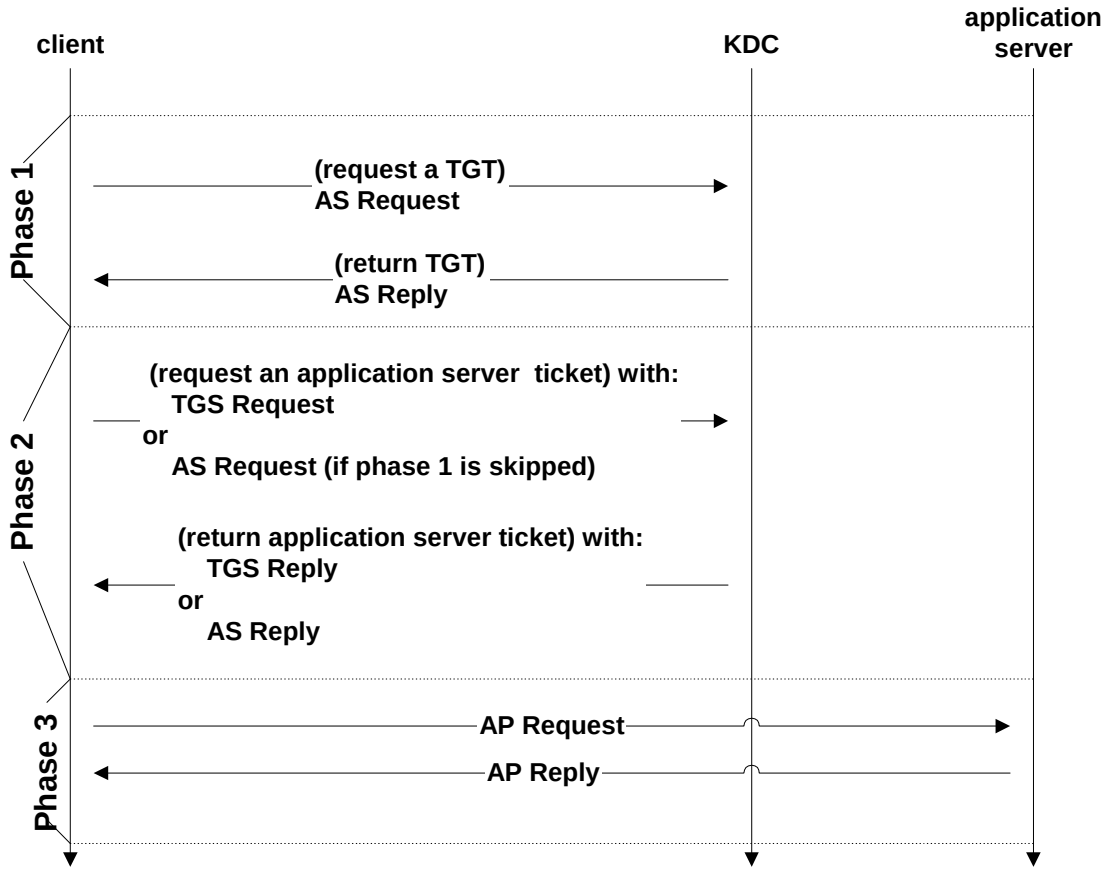


Figure 4. Kerberos-Based Key Management for IPsec

During the AS Request / AS Reply exchange (that can occur in either phase 1 or phase 2), the client and the KDC perform mutual authentication. In standard Kerberos, a client key that is shared with the KDC is used for this authentication (see Section 6.4.4). The same AS Request / AS Reply exchange may also be authenticated with digital signatures and certificates when the PKINIT public key extension is used (see Section 6.4.2). Both the TGT and the Application Server tickets used within PacketCable have a relatively long lifetime (days or weeks). This is acceptable as 3-DES, a reasonably strong symmetric algorithm, is required by PacketCable.

PacketCable utilizes the concept of a TGT (Ticket Granting Ticket), used to authenticate subsequent requests for Application Server tickets. The use of a TGT has two main advantages:

- It limits the exposure of the relatively long-term client key (that is in some cases reused as the service key). This consideration does not apply to clients that use PKINIT.
- It reduces the number of public key operations that are required for PKINIT clients.

The Application Server ticket contains a symmetric session key, which MUST be used in phase 3 to establish a set of keys for the IPsec ESP protocol. The keys used by IPsec MUST expire after a configurable time-out period (e.g., 10 minutes). Normally, the same Application Server ticket SHOULD be used to automatically establish a new IPsec SA. However, there are instances where it is desirable to drop IPsec sessions after a Security Association time out and establish them on-demand later. This allows for improved system scalability, since an application server (e.g., CMS) does not need to maintain a SA for every client (e.g., MTA) that it controls. It is also possible that a group of application servers (e.g., CMS cluster) may control the same subset of clients (e.g., MTAs) for load balancing. In this case, the MTA is not

required to maintain a SA with each CMS in that group. This section provides specifications for how to automatically establish a new IPsec SA right before an expiration of the old one and how to establish IPsec SAs on-demand, when a signaling message needs to be sent.

PacketCable also utilizes the Kerberos protocol to establish SNMPv3 keys between the MTAs and the Provisioning Server. Kerberized SNMPv3 key management is very similar to the Kerberized IPsec key management and consists of the same phases that were explained above for Kerberized IPsec. Each MTA again utilizes the PKINIT extension to Kerberos to authenticate itself to the KDC with X.509 certificates.

Once an MTA obtains its service ticket for the Provisioning Server, it utilizes the same protocol that is used for Kerberized IPsec to authenticate itself to the Provisioning Server and to generate SNMPv3 keys. The key management protocol is specified to allow application-specific data that has different profiles for SNMPv3 and IPsec. The only exception is the Rekey exchange that is specified for IPsec in order to optimize the MTA hand-off between the members of a CMS cluster. The Rekey exchange is not utilized for SNMPv3 key management.

A recipient of any Kerberos message that doesn't fully comply with the PacketCable requirements MUST reject the message.

6.4.1.1 Kerberos Ticket Storage

Kerberos clients that store tickets in persistent storage will be able to re-use the same Kerberos ticket after a reboot. In the event that PKINIT is used, this avoids the need to perform public key operations.

A Kerberos client MUST NOT obtain a new TGT upon reboot if it possesses a valid service ticket.

An MTA MUST store the Provisioning Server service ticket in persistent storage. An MTA MUST be capable of storing a minimum of pktcMtaDevEndPntCount+1 CMS service tickets in persistent storage, where pktcMtaDevEndPntCount is the MIB object specifying the number of physical endpoints on the MTA. An MTA MUST store all CMS service tickets that correspond to active endpoints. This means that an MTA that reaches the maximum number of CMS service tickets that can be stored in persistent storage will not over-write CMS service tickets that correspond to active endpoints.

Kerberos clients other than MTAs SHOULD retain service tickets in persistent storage.

Note that Kerberos clients will need to store additional information in order to use and validate the ticket, such as the session key information, the client IP address, and the ticket validity period. Refer to Section 7.1 for additional information on reusing stored tickets.

6.4.2 PKINIT Exchange

The diagram below illustrates how a client may use PKINIT to either obtain a TGT (phase 1) or a Kerberos ticket for an Application Server (phase 2).

The PKINIT Request is carried as a Kerberos pre-authenticator field inside an AS Request and the PKINIT Reply is a pre-authenticator inside the AS Reply. The syntax of the Kerberos AS Request / Reply messages and how pre-authenticators plug in is specified in Appendix B.

In this section, the PKINIT client is referred to as an MTA, as it is currently the only PacketCable element that authenticates itself to the KDC with the PKINIT protocol. If in the future other PacketCable elements will also utilize the PKINIT protocol, the same specifications will apply. PacketCable use of the AS Request / AS Reply exchange without PKINIT is covered in Section 6.4.3.

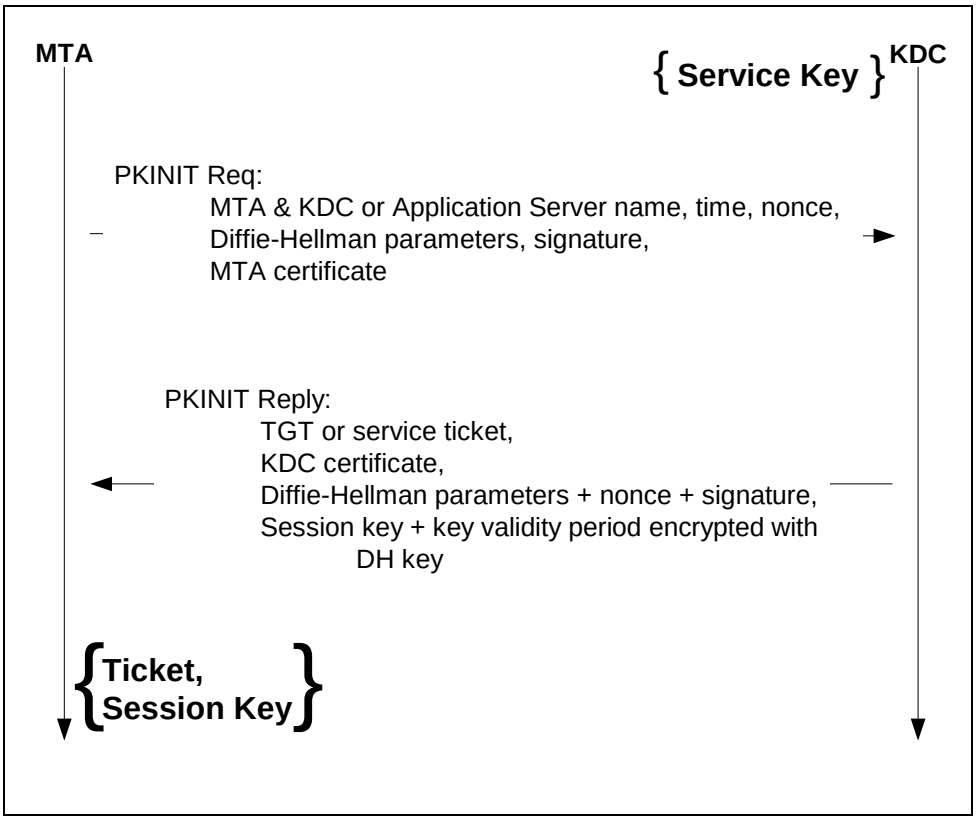


Figure 5. PKINIT Exchange

The above diagram lists several important parameters in the PKINIT Request and Reply messages. These parameters are:

PKINIT Request

- MTA (Kerberos principal) name – found in the KDC-REQ-BODY Kerberos structure (see Appendix B). For the format used in PacketCable, see Section 6.4.7.
- KDC or Application Server (Kerberos principal) name – found in the KDC-REQ-BODY Kerberos structure (see Appendix B). For the format used in PacketCable, see Section 6.4.6.
- Time – found in the PKAuthenticator structure, specified by PKINIT (Appendix C).
- Nonce - found in the PKAuthenticator structure, specified by PKINIT (Appendix C). There is also a second nonce in the KDC-REQ-BODY Kerberos structure.
- Diffie-Hellman parameters, signature and MTA certificate – these are all specified by PKINIT (Appendix C) and their use in PacketCable is specified in Section 6.4.2.1.1. Appendix G provides details of the first and second Oakley groups.

PKINIT Reply

- TGT or Application Server Ticket – found in the KDC-REP Kerberos structure (see Appendix B).
- KDC Certificate, Diffie-Hellman parameters, signature – these are all specified by PKINIT (see Appendix C) and their use in PacketCable is specified in Section 6.4.2.1.2. Appendix G provides details of the first and second Oakley groups.
- Nonce – found in the KdcDHKeyInfo structure, specified by PKINIT (Appendix C). This nonce must be the same as the one found in the PKAuthenticator structure of the PKINIT

Request. There is another nonce in EncKDCRepPart Kerberos structure (see Appendix B). This nonce must be the same as the one found in the KDC-REQ-BODY of the PKINIT Request.

- Session key, key validity period – found in the EncKDCRepPart Kerberos structure (see Appendix B).

In this diagram, the PKINIT exchange is performed at long intervals, in order to obtain an (intermediate) symmetric session key. This session key is shared between the MTA and the server via the server's ticket, where the application server may be the KDC (in which case the ticket is the TGT).

6.4.2.1 PKINIT Profile for PacketCable

A particular MTA implementation MUST utilize the PKINIT exchange to either obtain Application Server tickets directly, or obtain a TGT first and then use the TGT to obtain Application Server tickets. An MTA implementation MAY also support both uses of PKINIT, where the decision to get a TGT first or not is local to the MTA and is dependent on a particular MTA implementation. On the other hand, the KDC MUST be capable of processing PKINIT requests for both a TGT and for Application Server tickets.

The PKINIT exchange occurs independent of the signaling protocol, based on the current Ticket Expiration Time ($Ticket_{EXP}$) and on the PKINIT Grace Period ($PKINIT_{GP}$). If the PKINIT client is an MTA and the ticket it currently possesses corresponds to the Provisioning Server in the MIB, a KDC for a REALM that currently exists in the REALM table, or a CMS that currently exists in the CMS table, the MTA MUST initiate the PKINIT exchange at the time: $Ticket_{EXP}$ - $PKINIT_{GP.}$ If the PKINIT client is an MTA and the ticket it currently possesses does not correspond to the Provisioning Server in the MIB, a KDC for a REALM that currently exists in the REALM table, or a CMS that currently exists in the CMS table, the MTA MUST NOT initiate a PKINIT exchange. On the interfaces where $PKINIT_{GP}$ is not defined, the MTA SHOULD perform PKINIT exchanges on-demand.

In the case where PKINIT is used to obtain an Application Server ticket directly, the use of the grace period accounts for a possible clock skew between the MTA and the CMS or other application server. If the MTA is late with the PKINIT exchange, it still has until $Ticket_{EXP}$ before the Application Server starts rejecting the ticket. Similarly, if PKINIT is used to obtain a TGT the grace period accounts for a possible clock skew between the MTA and the KDC.

The PKINIT exchange stops after the MTA obtains a new ticket, and therefore does not affect existing security parameters between the MTA and the CMS or other application server. Synchronizing the PKINIT exchange with the AP Request/Reply exchange is not required as long as the AS Request/Reply exchange results in a valid, non-expired Kerberos ticket.

The PKINIT Request/Reply messages contain public key certificates, which make them longer than a normal size of a UDP packet. In this case, large UDP packets MUST be sent using IP fragmentation.

A KDC server should be implemented on a separate host, independent of the Application Server. This would mean, that frequent PKINIT operations from some MTAs will not affect the performance of any of the application servers or the performance of those MTAs that do not require frequent PKINIT exchanges.

Kerberos Tickets MUST NOT be issued for a period of time that is longer than 7 days. The MTA clock MUST NOT drift more than 2.5 minutes within that period (7 days). The PKINIT Grace Period $PKINIT_{GP}$ MUST be at least 15 minutes.

6.4.2.1.1 PKINIT Request

The PKINIT Request message (PA-PK-AS-REQ) in Appendix C is defined as:

```
PA-PK-AS-REQ ::= SEQUENCE {
  signedAuthPack    [0] ContentInfo,
  trustedCertifiers [1] SEQUENCE OF TrustedCas OPTIONAL,
  kdcCert           [2] IssuerAndSerialNumber OPTIONAL,
  encryptionCert    [3] IssuerAndSerialNumber OPTIONAL
}
```

The following fields MUST be present in PA-PK-AS-REQ for PacketCable (and all other fields MUST NOT be present):

- signedAuthPack – a signed authenticator field, needed to authenticate the client. It is defined in Cryptographic Message Syntax, identified by the SignedData OID:{iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2}. SignedData is defined as:

```
SignedData ::= SEQUENCE {
  version              CMSVersion,
  digestAlgorithms     DigestAlgorithmIdentifiers,
  encapContentInfo     EncapsulatedContentInfo,
  certificates     [0] IMPLICIT CertificateSet OPTIONAL,
  crls             [1] IMPLICIT CertificateRevocationLists
                       OPTIONAL,
  signerInfos          SignerInfos
}
```

  - digestAlgorithms - for now MUST contain an algorithm identifier for SHA-1. Other digest algorithms may optionally be supported in the future.
  - encapContentInfo – is of type EncapsulatedContentInfo that is defined by Cryptographic Message Syntax as:

```
EncapsulatedContentInfo ::= SEQUENCE {
  eContentType     ContentType,
  eContent     [0] EXPLICIT OCTET STRING OPTIONAL
}
```

    Here eContentType indicates the type of data and for PKINIT must be set to: {iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) pkauthdata(1)}

    eContent is a data structure of type AuthPack encoded inside an OCTET STRING:

```
AuthPack ::= SEQUENCE {
  pkAuthenticator   [0] PKAuthenticator,
  clientPublicValue [1] SubjectPublicKeyInfo OPTIONAL
}
```

    The optional clientPublicValue parameter inside the AuthPack MUST always be present for PacketCable. (This parameter specifies the client's Diffie-Hellman public value.)

```
PKAuthenticator ::= SEQUENCE {
  cusec      [0] INTEGER,
                 -- for replay prevention as in RFC1510
  ctime      [1] KerberosTime,
                 -- for replay prevention as in RFC1510
  nonce      [2] INTEGER,
                 -- zero only if client will accept
                 -- cached DH parameters from KDC;
                 -- must be non-zero otherwise
  pachecksum [3] Checksum
                 -- Checksum over KDC-REQ-BODY
                 -- Defined by Kerberos spec
}
```

    The pachecksum field MUST use the Kerberos checksum type rsa-md5, a plain MD5 checksum over the KDC-REQ-BODY.

    The nonce field MUST be non-zero, indicating that the client does not support the caching of Diffie-Hellman values and their expiration.

- certificates - required by PacketCable. This field MUST contain an MTA Device Certificate and an MTA Manufacturer Certificate. This field MUST NOT contain any other certificates. All PacketCable certificates are X.509 certificates for RSA Public keys as specified in Section 8.
- crls – MUST NOT be filled in by the MTA.
- signerInfos – MUST be a set with exactly one member that holds the MTA signature. This signature is a part of a SignerInfo data structure defined within the Cryptographic Message Syntax. All optional fields in this data structure MUST NOT be used in PacketCable. The digestAlgorithm MUST be set to SHA-1:

  {iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 26}

  and the signatureAlgorithm MUST be set to rsaEncryption:

  {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1}

PKINIT allows an Ephemeral-Ephemeral Diffie-Hellman exchange as part of the PKINIT Request/Reply sequence. (Ephemeral-Ephemeral means that both parties during each exchange randomly generate the Diffie-Hellman private exponents.) The Kerberos session key is returned to the MTA in the PKINIT Reply, encrypted with a secret that is derived from the Diffie-Hellman exchange. Within PacketCable, the Ephemeral-Ephemeral Diffie-Hellman MUST be supported.

The IKE specification in [24] defines Diffie-Hellman parameters as Oakley groups. Within the PacketCable PKINIT profile the 2nd Oakley group MUST be supported and the 1st Oakley group MAY also be supported. Appendix G provides details of the first and second Oakley groups.

When generating Diffie-Hellman private keys, a device MUST generate a key of length at least 144 bits when the first Oakley group is used and MUST generate a key of length at least 164 bits when the second Oakley group is used.

For further details of PKINIT, please refer to Appendix C.

Additionally, PKINIT supports a Static-Ephemeral Diffie-Hellman exchange, where the client is required to possess a Diffie-Hellman certificate in addition to an RSA certificate. This mode MUST NOT be used within PacketCable.

PKINIT also allows a single client RSA key to be used both for digital signatures and for encryption - wrapping the Kerberos session key in the PKINIT Reply. This mode MUST NOT be used within PacketCable.

PKINIT has an additional option for a client to use two separate RSA keys – one for digital signatures and one for encryption. This mode MUST NOT be used within PacketCable.

Upon receipt of a PA-PK-AS-REQ, the KDC MUST:

1. check the validity of the certificate chain (MTA Device Certificate, MTA Manufacturer Certificate, MTA Root Certificate)
2. check the validity of the signature in the (single) SignerInfo field
3. check the validity of the checksum in the PKAuthenticator

6.4.2.1.2 PKINIT Reply

The PKINIT Reply message (PA-PK-AS-REP) in Appendix C is defined as follows:

```
PA-PK-AS-REP ::= CHOICE {
  dhSignedData [0] ContentInfo,
  encKeyPack   [1] ContentInfo
}
```

PacketCable MUST use only the dhSignedData choice, which is needed for a Diffie-Hellman exchange.

The value of the Kerberos session key is not present in PA-PK-AS-REP. It is found in the encrypted portion of the AS Reply message that is specified in Appendix B. The AS Reply is encrypted with 3-DES CBC, with a Kerberos etype value of des3-cbc-md5 (see Section 6.4.2.2). Other encryption types may be supported in the future.

The client MUST use PA-PK-AS-REP to determine the encryption key used on the AS Reply. This PKINIT Reply contains the KDC's Diffie-Hellman public value that is used to generate a shared secret (part of the key agreement). This shared secret is used to encrypt/decrypt the private part of the AS Reply.

- dhSignedData - dhSignedData is identified by the SignedData oid: {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2}. Within SignedData (specified in Section 6.4.2.1.1):
    - digestAlgorithms - for now MUST contain an algorithm identifier for SHA-1. Other digest algorithms may optionally be supported in the future.
    - encapContentInfo – is of type pkdhkeydata, where eContentType contains the following OID value: {iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) pkdhkeydata(2)}

      eContent is of type KdcDHKeyInfo (encoded inside an OCTET STRING):

```
KdcDHKeyInfo ::= SEQUENCE {
                 -- used only when utilizing Diffie-Hellman
  subjectPublicKey [0] BIT STRING,
                       -- Equals public exponent (g^a mod p)
                       -- INTEGER encoded as payload of
                       -- BIT STRING
  nonce            [1] INTEGER,
                       -- Binds response to the request
                       -- Exception: Set to zero when KDC
                       -- is using a cached DH value
  dhKeyExpiration  [2] KerberosTime OPTIONAL
                       -- Expiration time for KDC's cached
                       -- DH value
}
```

        - The nonce MUST be the same nonce that was passed in by the client in the PKINIT Request.
        - The subjectPublicKey MUST be the Diffie-Hellman public value generated by the KDC. The Diffie-Hellman-derived key is used to directly encrypt part of the AS Reply. The requirements on the length of the Diffie-Hellman private exponent are as defined in Section 6.4.2.1.1.
        - The dhKeyExpiration MUST not be present as caching of Diffie-Hellman values is not permitted.
    - certificates – required by PacketCable. This field MUST contain a KDC certificate. If a Local System CA issued the KDC certificate, then the corresponding Local System CA Certificate MUST also be present. The Service Provider CA Certificate MUST also be present in this field. This field MAY contain the Service Provider Root CA certificate (refer to Section 8.2.1 for validating the Service Provider Root CA certificate if it is included in the PKINIT Reply). This field MUST NOT contain any other certificates. If the MTA is configured with a specific service provider name, it MUST verify that the Service Provider name is identical to the value of the OrganizationName attribute in the subjectName of the Service Provider certificate. If the Local System Certificate is present, then the MTA MUST verify that the Service Provider name is identical to the value of the OrganizationName attribute in the subjectName of the Local System Certificate. In addition to standard certificate verification rules specified in RFC 2459, an MTA MUST verify that the KDC certificate includes a subjectAltName extension in the format specified in Section 8.2.3.4.1. The MTA MUST verify that the extension contains a valid KDC principal name and that the KDC realm in this

extension is identical to the server realm name in the encrypted portion of the AS Reply message (EncKDCRepPart).

- crls – this optional field MAY be filled in by the KDC.
- signerInfos – MUST be a set with exactly one member that holds the KDC signature. This signature is a part of a SignerInfo data structure defined within the Cryptographic Message Syntax. All optional fields in this data structure MUST NOT be used in PacketCable. The digestAlgorithm MUST be set to SHA-1:

  {iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 26}

  the signatureAlgorithm MUST be set to rsaEncryption:

  {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1}

Upon receipt of a PA-PK-AS-REP, the client MUST:

1. check the value of the nonce in the eContent field
2. check the validity of the KDC certificate
3. check the validity of the signature in the SignerInfo field

6.4.2.1.2.1 PKINIT Error Messages

In the case that a PKINIT Request is rejected, instead of a PKINIT Reply the KDC MUST return a Kerberos error message of type KRB_ERROR, as defined in Appendix C. Any error code that is defined in Appendix C for PKINIT MAY be returned.

- The KRB_ERROR MUST use typed-data of REQ-NONCE to bind the error message to the nonce from the KDC-REQ-BODY portion of the AS-REQ message. This error message MUST NOT include the optional e-cksum member that would contain a keyed checksum of the error reply. The use of this field is not possible during the PKINIT exchange, since the client and the KDC do not share a symmetric key.

When a client receives an error message from the KDC, in some cases this specification calls for the client to take some recovery steps and then send a new AS Request or TGS Request. When a client is responding to an error message, it is not a retry and MUST NOT be considered to be part of the client's back-off and retry procedure specified in Section 6.4.8. The client MUST reset its timers accordingly, to reflect that the new request in response to an error message is not a retry.

Although this specification calls for a KDC to return some specific error codes under certain error conditions, in the case when a KDC is repeatedly getting the same error from the same client IP address, it MAY at some point choose to stop sending back any further replies (errors or otherwise) to this client.

6.4.2.1.2.1.1 Clock Skew Error

When the KDC clock and the client clock are off by more than the limit for a clock skew, an error code KRB_AP_ERR_SKEW MUST be returned. The value for the maximum clock skew allowed by the KDC MUST NOT exceed 5 minutes. The optional client's time in the KRB_ERROR MUST be filled out, and the client MUST compute the difference (in seconds) between the two clocks based upon the client and server time contained in the KRB_ERROR message. The client SHOULD store this clock difference in non-volatile memory and MUST use it to adjust Kerberos timestamps in subsequent KDC request messages (AS Request and TGS Request) by adding the clock skew to its local clock value each time. The client MUST maintain a separate clock skew value for each realm. The clock skew values are intended for uses only within the Kerberos protocol and SHOULD NOT otherwise affect the value of the local clock (since a clock skew is likely to vary from realm to realm).

In the case that a KDC request fails due to a clock skew error, a client MUST immediately retry after adjusting the Kerberos timestamp inside the KDC Request message.

In addition, the MTA MUST validate the time offset returned in the clock skew error, to make sure that it does not exceed a maximum allowable amount. This maximum time offset MUST NOT exceed 1 hour.

This MTA check against a maximum time offset protects against an attack in which a rogue KDC attempts to fool an MTA into accepting an expired KDC certificate.

6.4.2.1.3 Pre-Authenticator for Provisioning Server Location

An AS Request sent by the MTA MUST include this PROV-SRV-LOCATION pre-authenticator that the KDC can use to locate the Provisioning Server.

The pre-authenticator type MUST be –1 (according to Appendix B, the negative type is used for application-specific pre-authenticators). Its ASN.1 encoding is specified as:

```
PROV-SRV-LOCATION ::= GeneralString
                      -- Provisioning Server's FQDN
```

6.4.2.2 Profile for the Kerberos AS Request / AS Reply Messages

As mentioned earlier, the PKINIT Request and Reply are pre-authenticator fields embedded into the AS Request / AS Reply messages. The PacketCable-specific PROV-SRV-LOCATION pre-authenticator MUST be used in combination with PKINIT. All other pre-authenticators MUST NOT be used in combination with PKINIT.

The optional fields from, enc-authorization-data, additional-tickets and rtime in the KDC-REQ-BODY MUST NOT be present in the AS Request. All other optional fields in the AS Request MAY be present for PacketCable. The client MUST NOT set any of the KDCOptions in the AS-REQUEST, except that the DISABLE-TRANSITED-CHECK option MAY be set.

The MTA MUST include its IP address in the optional addresses field of the KDC-REQ-BODY. The KDC MUST verify that the addresses field in the KDC-REQ-BODY contains exactly one IP address and that it is identical to the IP address in the IP header of the AS Request. After the KDC validates the addresses field, it MUST include it in the caddr fields of the issued ticket and the AS Reply. The KDC MUST reject an AS Request that does not include the MTA's IP address. In this case the KDC MUST return a KDC_ERR_POLICY error code.

If a KDC receives an AS-REQ message in which any of the KDCOptions are set, except for the DISABLE-TRANSITED-CHECK option, the KDC MUST return an error with the error code KDC_ERR_POLICY.

In the AS Reply, key-expiration, starttime and renew-till optional fields MUST NOT be present. The session key contained in the AS-REPLY (which MUST be identical to the session key in the ticket) MUST be etype des3-cbc-md5.

The encrypted part of the AS Reply is of the type EncryptedData. The ASN.1 definition of EncryptedData that is used inside multiple Kerberos objects is missing from the Kerberos revisions IETF draft in Appendix B. In all cases, EncryptedData MUST be DER-encoded with EXPLICIT tags as the following ASN.1 structure:

```
EncryptedData   ::= SEQUENCE {
       etype   [0] INTEGER,           -- EncryptionType
       kvno    [1] INTEGER OPTIONAL,  -- service key
                                          -- version number
       cipher  [2] OCTET STRING       -- ciphertext
}
```

When EncryptedData contains ciphertext that is encrypted with a service key, the 'kvno' element MUST be present and MUST identify the version of the service key that was used to encrypt the data. When EncryptedData contains ciphertext that is encrypted with a Kerberos session key or with a reply key derived from a PKINIT pre-authenticator, the 'kvno' element MUST NOT be present. This is the case for the encrypted portion of the AS Reply.

The encryption type for an encrypted portion of the AS Reply MUST be set to des3-cbc-md5. In order to generate the value of the 'cipher' element of the EncryptedData, the following data MUST be concatenated and processed in the following sequence before being encrypted with 3-DES CBC, IV=0:

- 8-byte random byte sequence, called a confounder
- An MD5 checksum, which is the MD5 hash of the concatenation of the three quantities (the confounder + sixteen NULL octets + the text to be encrypted [not including any padding])
- AS Reply part that is to be encrypted
- Random padding up to a multiple of 8

Upon receipt of an AS-REPLY, the client MUST check the validity of the checksum in the encrypted portion of the AS-REPLY.

6.4.2.3 Profile for Kerberos Tickets

In Kerberos Tickets, authorization-data, starttime and renew-till optional fields MUST NOT be present. The optional caddr field MUST be present when requested in an AS-REQUEST or when present in a TGT of a TGS Request (see Appendix B). The only ticket flags that are supported within PacketCable are the INITIAL, PRE-AUTHENT and TRANSITED-POLICY-CHECKED flags. If the KDC receives any request that would otherwise cause it to set any other flag, it MUST return an error with the error code KDC_ERR_POLICY. The KDC MUST NOT generate tickets with any other flags set. The session key contained in the ticket (which MUST be identical to the session key in the AS-REPLY) MUST be etype des3-cbc-md5. Since the transited encoding information normally required by PKINIT (see Appendix B, 3.3.3.2) is not used in PacketCable, a KDC MAY choose to leave as a null string the 'contents' field of the TransitedEncoding portion of a ticket issued in response to a PKINIT request.

The encrypted part of the Kerberos ticket MUST be encrypted with the encryption type set to des3-cbc-md5, using the same procedure as described in the above Section 6.4.2.2.

Upon receipt of a ticket for a service, the server MUST:

1. check the validity of the checksum in the encrypted portion of the ticket
2. check that the ticket has not expired

Currently, all the service keys are pre-shared using an out-of-band mechanism between the KDC and the device providing the service. In the future, PacketCable may support a method that does not require these keys to be pre-shared.

6.4.3 Symmetric Key AS Request / AS Reply Exchange

In PacketCable, a Kerberos client MAY use standard symmetric-key authentication (with a client key) during the AS Request / AS Reply exchange. Also, in PacketCable, a client not utilizing PKINIT is, at the same time, an Application Server for which other clients might obtain tickets. This means that a PacketCable entity may utilize the same symmetric key for both client authentication and for decrypting its service tickets.

The Kerberos AS Request / AS Reply exchange, in general, is allowed to occur with no client authentication. The client, in those cases, would authenticate itself later by proving that it is able to decrypt the AS Reply with its symmetric key and make use of the session key.

Such use of Kerberos is not acceptable within PacketCable. This approach would allow a rogue client to continuously generate AS Requests on behalf of other clients and receive the corresponding AS Replies. Although this rogue client would be unable to decrypt each AS Reply, it will know some of the fields that it should contain. This, and the availability of the matching encrypted AS Replies, would aid an attacker in the discovery of another client's key with cryptanalysis.

Therefore, PacketCable requires that whenever an AS Request is not using a PKINIT preauthenticator, it MUST instead use a different preauthenticator, of type PA-ENC-TS-ENC. This preauthenticator is specified as:

```
PA-ENC-TS-ENC ::= SEQUENCE {
  patimestamp [0] KerberosTime,
                  -- client's time
  pausec      [1] INTEGER OPTIONAL,
  pachecksum  [2] CheckSum OPTIONAL
                  -- keyed checksum of
                  -- KDC-REQ-BODY
}
```

The PA-ENC-TS-ENC preauthenticator MUST be encrypted with the client key using the encryption type des3-cbc-md5, as described in Section 6.4.2.2. All optional fields inside PA-ENC-TS-ENC MUST be present for PacketCable. The pachecksum field MUST be a keyed checksum of type rsa-md5-des3. The checksum MUST be keyed with the client key. The checksum MUST be validated by the KDC.

The encrypted timestamp is used by the KDC to authenticate the client. At the same time, the timestamp inside this preauthenticator is used to prevent replays. The KDC checks for replays upon the receipt of this preauthenticator; this is similar to the checking performed by an Application Server upon receipt of an AP Request message.

If the timestamp in the PA-ENC-TS-ENC preauthenticator differs from the current KDC time by more than the acceptable clock skew then KDC MUST reply with a clock skew error message. The client MUST respond to this error message as specified in Section 6.4.2.1.2.1.1

If the realm, target server name (e.g., the name of the KDC), along with the client name, time and microsecond fields from the PA-ENC-TS-ENC preauthenticator match any recently-seen such tuples, the KRB_AP_ERR_REPEAT error MUST be returned. The KDC MUST remember any such preauthenticator presented within acceptable clock skew period, so that a replay attempt is guaranteed to fail.

If the Application Server loses track of any authenticator presented within the acceptable clock skew period, it MUST reject all requests until the acceptable clock skew interval has passed.

Symmetric-key AS Request / AS Reply exchange is illustrated in the following figure:

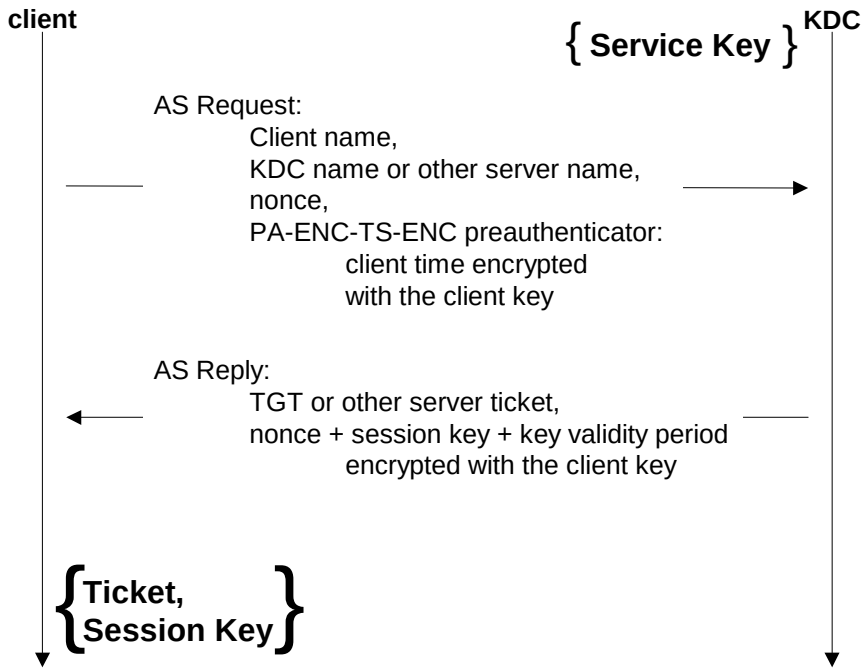


Figure 6. Symmetric-Key AS Request / AS Reply Exchange

6.4.3.1 Profile for the Symmetric Key AS Request / AS Reply Exchanges

The content of the AS Request / AS Reply messages is the same as in the case of the PKINIT preauthentication (see Section 6.4.2.1) with the exception of the type of the preauthenticator that is used.

In general, clients using a symmetric-key form of the AS Request / AS Reply exchange are not required to always possess a valid TGT or a valid Application Server ticket. A client MAY obtain both a TGT and Application Server tickets on-demand, as they are needed for the key management with the Application Server.

However, there may be cases where a client is required to quickly switch between servers for load balancing and the additional symmetric-key exchanges with the KDC are undesirable. In those cases, a client MAY be optimized to obtain tickets in advance, so that the key management would take only a single roundtrip (AP Request / AP Reply exchange.)

In the case that the KDC rejects the AS Request, it returns a KRB_ERROR message instead of the AS Reply, as specified in Appendix B. The KRB_ERROR MUST use typed-data of REQ-NONCE to bind the error message to the nonce from the AS-REQ message. This error message MUST include the optional e-cksum member that would contain an rsa-md5-des3 keyed checksum of the error reply, unless pre-authentication failed to prove knowledge of the shared symmetric key in which case the e-cksum MUST NOT be used.

The rsa-md5-des3 checksum MUST be computed as follows:

1. prepend the message with an 8-byte random byte sequence, called a confounder
2. take an MD5 hash of the result of step 1
3. prepend the hash with the same 8-byte confounder
4. take the 3DES session key from the ticket and XOR each byte with F0
5. use 3DES in CBC mode to encrypt the result of step 3, using the key in step 4 and with IV(initialization vector)=0

Once a client receives an AS Reply, it SHOULD save both the obtained ticket and the session key information (found in the enc-part member of the reply) in non-volatile memory. Thus, the client will be able to re-use the same Kerberos ticket after a reboot, avoiding the need to perform the AS Request again.

Kerberos Tickets MUST NOT be issued for a period of time that is longer than 7 days (same as for PKINIT exchanges).

Upon receipt of a KRB_ERROR that contains an e-cksum field, the recipient MUST verify the validity of the checksum.

6.4.4 Kerberos TGS Request / TGS Reply Exchange

In the cases where a client obtained a TGT, that TGT is then used in the TGS Request / TGS Reply exchange to obtain a specific Application Server ticket. This is part of the Kerberos standard, as it is specified in Appendix B.

A TGS Request includes a KRB_AP_REQ data structure (the same structure used in an AP Request: see Section 6.4.4.1). This data structure contains the TGT as well as an authenticator that is used by the client to prove the possession of the corresponding session key. The TGS Reply has the same format as an AS Reply, except that it is encrypted using a different key – the session key from the TGT.

Figure 7 illustrates the TGS Request / TGS Reply exchange:

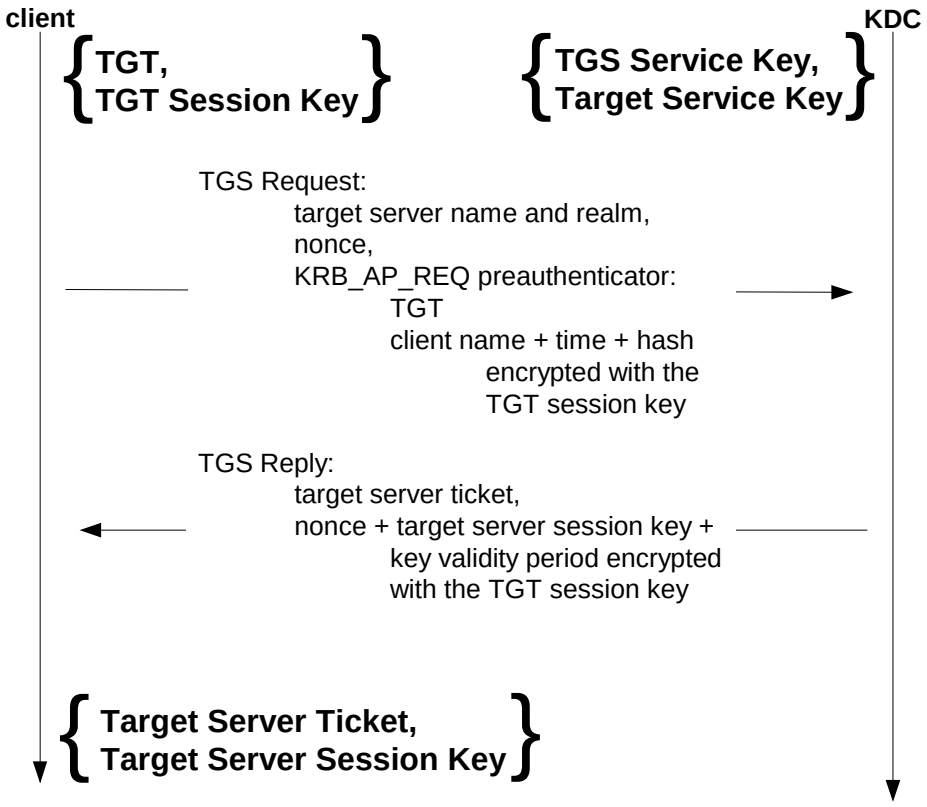


Figure 7. Kerberos TGS Request / TGS Reply Exchange

Figure 7 lists several important parameters in the TGS Request and Reply messages. These parameters are:

- TGS Request
  - Target server (principal) name and realm, nonce – found in the KDC-REQ-BODY Kerberos structure (see Appendix B)
  - TGS preauthenticator - found in the KDC-REQ Kerberos structure, inside the padata field (see Appendix B). The preauthenticator type in this case is PA-TGS-REQ.
  - KRB_AP_REQ – the value of the preauthenticator of type PA-TGS-REQ
  - TGT – inside the KRB_AP_REQ
  - Client name, time – inside the Kerberos Authenticator structure, which is embedded in an encrypted form in the KRB_AP_REQ
- TGS Reply
  - Target server ticket – found in the KDC-REP Kerberos structure (see Appendix B)
  - Target server session key, nonce, key validity period – found in the EncKDCRepPart Kerberos structure (see Appendix B)

In general, the TGS Request / Reply exchange may be performed on-demand - whenever an Application Server ticket is needed to establish Security Parameters. If the client is an MTA and a ticket it currently possesses corresponds to the Provisioning Server in the MIB or a CMS that currently exists in the CMS table, it MUST initiate the TGS Request / Reply exchange at the time: $Ticket_{EXP} - TGS_{GP}$. Here, $Ticket_{EXP}$ is the expiration time of the current Application Server ticket and TGS_{GP} is the TGS Grace Period. If the

client is an MTA and the ticket it currently possesses does not correspond to the Provisioning Server in the MIB or a CMS that currently exists in the CMS table, the MTA MUST NOT initiate a PKINIT exchange.

The validity of the Application Server tickets MUST NOT extend beyond the expiration time of the TGT that was used to obtain the server ticket.

6.4.4.1 TGS Request Profile

The optional padata element in the KDC-REQ data structure MUST consist of exactly one element – a preauthenticator of type PA-TGS-REQ. The value of this preauthenticator is the KRB_AP_REQ data structure. Within KRB_AP_REQ:

1. Options in the ap-options field MUST NOT be present.
2. The ticket is the TGT.
3. The encrypted authenticator MUST contain the checksum field – an MD5 checksum of the ASN.1 encoding of the KDC-REQ-BODY data structure. It MUST NOT contain any other optional fields.
4. The authenticator MUST be encrypted using 3-DES CBC with the following Kerberos etype value des3-cbc-md5 as specified in Section 6.4.2.2.

The optional fields from, enc-authorization-data, additional-tickets and rtime in the KDC-REQ-BODY MUST NOT be present in the TGS Request. The optional field cname SHOULD NOT be present. All other optional fields in the TGS Request MAY be present for PacketCable. The KDC MUST reject a TGT that has any ticket flags set, apart from the flags INITIAL, PRE-AUTHENT or TRANSITED-POLICY-CHECKED. If the KDC receives any request that would otherwise cause it to set any flag in the service ticket, apart from the PRE-AUTHENT and TRANSIT-POLICY-CHECKED flags, it MUST return an error with the error code KDC_ERR_POLICY. The KDC MUST NOT generate TGT-based service tickets with any other flags set.

If the TGT contains a caddr field, the KDC MUST verify that it is a single IP address and that it is identical to the IP address in the IP header of the TGS Request. The KDC MUST reject TGS Requests from an MTA with a TGT that does not include the MTA's IP address, returning a KDC_ERR_POLICY error code (refer to Section 6.4.4.3).

Upon receipt of a TGS Request, the KDC MUST:

1. check the validity of the TGT;
2. check the validity of the checksum in the authenticator.

6.4.4.2 TGS Reply Profile

In the TGS Reply, key-expiration, starttime and renew-till optional fields MUST NOT be present. The encrypted part of the TGS Reply MUST be encrypted with the encryption type set to des3-cbc-md5, using the same procedure as described in Section 6.4.2.2.

Upon receipt of a TGS Reply, the client MUST:

1. use the value of the nonce to bind the reply to the corresponding TGS Request;
2. check the validity of the checksum in the encrypted portion of the TGS Reply.

6.4.4.3 Error Reply

If the KDC is able to successfully parse the TGS Request and the TGT that is inside of it, but the TGS Request is rejected, it MUST return a Kerberos error message of type KRB_ERROR, as defined in Appendix B. The error message MUST include the optional e-cksum member, which is the keyed hash over the KRB_ERROR message. The checksum type MUST be rsa-md5-des3, calculated using the procedure described in Section 6.4.3.1.

The KRB_ERROR MUST also include typed-data of REQ-NONCE to bind the error message to the nonce from the TGS-REQ message.

Upon receipt of a KRB_ERROR, the client MUST check the validity of the checksum.

6.4.5 Kerberos Server Locations and Naming Conventions

6.4.5.1 Kerberos Realms

A realm name MAY use the same syntax as a domain name, however Kerberos Realms MUST be in all capitals. For a full specification of Kerberos realms, refer to Appendix B.

6.4.5.2 KDC

Kerberos principal identifier for the local KDC when it is in a role of issuing tickets is always: krbtgt/<realm>@<realm>, where <realm> is the Kerberos realm corresponding to the particular PacketCable zone. This is the service name listed inside a TGT.

A Kerberos client MUST query KDC FQDNs for a particular realm name using DNS SRV records, as specified in [39] and as shown below:

<Service Name>.<Protocol>.<Name> TTL Class SRV Priority Weight Port Target

Where:

- The Service Name for Kerberos in PacketCable MUST be "_kerberos".
- The Protocol for Kerberos in PacketCable MUST be "_udp".
- The Name MUST be the Kerberos realm name that this record corresponds to.
- TTL, Class, SRV, Priority, Weight, Port, and Target have the standard meaning as defined in [39].

For example, assume the presence of a realm, PACKETCABLE.COM, with two KDCs: kdc1.packetcable.com and kdc2.packetcable.com. These KDCs have different priorities. The DNS SRV records in this case would be:

_kerberos._udp.PACKETCABLE.COM. 86400 IN SRV 0 0 88 kdc1.packetcable.com.

_kerberos._udp.PACKETCABLE.COM. 86400 IN SRV 1 0 88 kdc2.packetcable.com.

To obtain records pertaining to the realm PACKETCABLE.COM, the MTA would send a DNS SRV request for:

_kerberos._udp.PACKETCABLE.COM

The client, upon receiving a response for a DNS SRV request, MUST consider the priority/weight as described in the algorithm in [39] and contact the servers in that order. A client MUST contact the next server based on priority/weight and so on, till all possible server FQDNs and the corresponding IPs are exhausted, if it fails to get a suitable response from the first server listed (refer to 6.4.8 for timeout procedures).

For example, after the above DNS SRV records are retrieved, the client will try kdc1.packetcable.com first, based on its priority. (Priority for kdc1.packetcable.com is 0, while priority for kdc2.packetcable.com is 1: a lower priority number means a higher priority.)

When a PacketCable KDC is requesting information from a Provisioning Server (e.g., the mapping of an MTA MAC address to its corresponding FQDN) it MUST use a principal name of type NT- PRINCIPAL (1) with a single component "kdcquery" (without quotes).

In an ASCII representation, the principal identifier is as follows:

kdcquery@<realm>

where <realm> is the Kerberos realm of the KDC.

6.4.5.3 CMS

A CMS Kerberos principal identifier MUST be constructed from the CMS FQDN as follows:

cms/<FQDN>@<realm>

where <FQDN> is the CMS's FQDN (in lower case) and <realm> is its Kerberos realm.

For example, a CMS with an FQDN 'iptel-cms1.company1.com' and with a realm name 'COMPANY1.COM' would have the principal identifier:

cms/iptel-cms1.company1.com@COMPANY1.COM

The Kerberos PrincipalName data structure (inside the Kerberos messages) is defined as follows:

```
PrincipalName ::= SEQUENCE {
  name-type   [0] INTEGER,
  name-string [1] SEQUENCE OF GeneralString
}
```

Within this data structure, name-type MUST be NT-SRV-HST (which has the value of 3 according to the Kerberos specification). The name-string element of the data structure MUST have exactly two components, where the first component has the string value "cms" (without the quotes) and the second component is the CMS's FQDN in lower case.

For the full syntax of Kerberos principal names, refer to Appendix B.

For the purpose of setting up an IPsec connection between the CMS and RKS, the first component of the CMS principal name MUST be of the form "cms:<ElementID", where the <ElementID> is described in Section 6.4.5.5.

In the case of a combined network element that integrates the functions of multiple logical elements within the PacketCable reference architecture (e.g., a single network element that provides both CMS and MGC functionality), the principal name may include all server functions as specified in Section 6.4.5.5.

6.4.5.4 Provisioning Server

When a PacketCable MTA Provisioning Server is acting in the role of an SNMP manager, it MUST use a principal name of type NT-SRV-HST (3) with the following two components:

1. "mtaprovsrvr" (without quotes)
2. the FQDN of the Provisioning Server (in lower case)

In ASCII representation, the Provisioning Server's principal identifier MUST be as follows:

mtaprovsrvr/<Prov Server FQDN>@<realm>

where <realm> is the Kerberos realm of the Provisioning Server.

When a PacketCable Provisioning Server is providing a service (to the KDC) that maps each MTA MAC address to its corresponding FQDN, it MUST use a principal name of type NT-SRV-HST (3) with the following two components:

1. "mtafqdnmap" (without quotes)
2. the FQDN of the Provisioning Server (in lower case)

In ASCII representation, the principal identifier MUST be as follows:

mtafqdnmap/<Prov Server FQDN>@<realm>

where <realm> is the Kerberos realm of the Provisioning Server.

6.4.5.5 Names of Other Kerberized Services

All Kerberized services within PacketCable, except for the KDC krbtgt service (see 6.4.5.2), MUST be assigned a service principal name of type KRB_NT_SRV_HST (Value=3), which has the following form according to the Kerberos specification:

<service name>/<FQDN>

This means that the first component of the service principal name is the service name in lower case, and the last is either an FQDN in lower-case or an IP address of the corresponding host. If a specific host has an assigned FQDN, its principal name includes an FQDN and not an IP address. When a KDC receives a ticket request for a service on this host with an IP address instead of an FQDN as the second component of the service principal name, the KDC MUST reject such a request.

When a KDC database contains a service with a principal name that has an IP address as the second component, all ticket requests for this service MUST use the same service principal name with the same IP address as the second component. When a KDC receives a ticket request for this service with an FQDN as the second component of the service principal name, the KDC MUST reject such a request. (This scenario could happen if a service principal is defined in the KDC database at the time when the corresponding host does not have an FQDN, and then later an FQDN for this host is defined as well.)

When an IP address is used, it MUST be formatted as follows:

[A.B.C.D]

where A, B, C and D are components of an IPv4 address expressed as decimal numbers. The components of an IP address MUST be separated by a period '.' and the IP address MUST be surrounded by square brackets.

The following is an example of a principal name based on an IP address:

df/[192.35.65.4]

Figure 3 shows a number of interfaces for which the necessary security is provided by IPsec. In addition to supplying the required key management using IKE with pre-shared keys, some vendors may choose to implement, and operators to deploy, a Kerberized key management scheme for these interfaces.

This specification requires that the RKS verifies billing event messages by ensuring that the Element ID contained in the message matches correctly the IP address at the far end of the IPsec Security Associations. In order to ensure that the RKS is able to maintain this mapping when Kerberized key management is used to generate the Security Associations, devices that communicate with the RKS include their Element ID in their principal name. This information is then passed to the RKS in the cname field of the ticket that the KDC issues; this ticket is passed to the RKS in the AP-REQ that is used to initiate the IPsec Security Associations.

The first component of the principal name for the various PacketCable devices MUST be as follows:

1. BP: bp[:<ElementID>]
2. CMTS: cmts[:<ElementID>]
3. DF: df[:<ElementID>]
4. MG: mg[:<ElementID>]
5. MGC: mgc[:<ElementID>]
6. MP: mp[:<ElementID>]
7. MPC: mpc[:<ElementID>]

8. RKS: rks[:<ElementID>]
9. SG: sg[:<ElementID>]

where:

<ElementID> is the identifier that appears in billing event messages and it MUST be included in a principal name of every server that is capable of generating event messages.

Element ID is defined as an 5-octet right-justified, space-padded ASCII-encoded numerical string [6]. When converting the Element ID for use in a principal name, any spaces MUST be converted to ASCII zeroes (0x48).

For example, a CMTS that has the Element ID " 311" will have a principal name whose first component is "cmts:00311". Similarly, a DF with no Element ID will have a principal name whose first component is "df".

Components that contain combined elements (such as a CMS with an integrated MGC) MUST indicate this in the principal name by including all component names, joined with the character "&", in the first component of the principal name. The following is an example of a principal name for a combined CMS and MGC with a single IP address:

cms:00210&mgc:00211/[192.35.65.4]

If the combined component uses a single ElementID, the principal name would be:

cms:00210&mgc:00210/[192.35.65.4]

6.4.6 MTA Principal Names

An MTA principal name MUST be of type NT-SRV-HST with exactly two components, where the first component MUST be the string "mta" (not including the quotes) and the second component MUST be the FQDN of the MTA:

mta/<MTA FQDN>

where <MTA FQDN> is the FQDN of the MTA in lower case.

For example, if an MTA FQDN is "mta12345.mso1.com" and its realm is "MSO1.COM", the principal identifier would be:

mta/mta12345.mso1.com@MSO1.COM

6.4.7 Mapping of MTA MAC Address to MTA FQDN

The MTA authenticates itself with the MTA Device Certificate in the AS Request, where the certificate contains the MTA MAC address but not its FQDN. In order to authenticate the MTA principal name (containing the FQDN), the KDC MUST map the MTA MAC address (from the MTA Device certificate) to the MTA FQDN, in order to verify the principal name in the AS Request.

The protocol for retrieving the MTA FQDNs is Kerberos-based. The Provisioning Server MUST listen for the request on UDP port 2246 and MUST return the response to the UDP port from which the request was transmitted on the client:

1. MTA FQDN Request – sent from the KDC to the Provisioning Server, containing the MTA MAC address and the hash of the MTA public key. This message consists of the Kerberos KRB_AP_REQ concatenated with KRB_SAFE.
2. MTA FQDN Reply – a reply to the KDC by the Provisioning Server, containing the MTA FQDN. This message consists of the Kerberos KRB_AP_REP concatenated with KRB_SAFE.
3. MTA FQDN Error Reply – an error reply in response to the MTA FQDN Request. This message is the Kerberos KRB_ERROR.

The format of each of these messages is specified in the subsections below.

6.4.7.1 MTA FQDN Request

The KDC MUST first verify the digital signature and certificate chain in the PKINIT Request, before sending out an MTA FQDN Request message to determine the MTA MAC address to FQDN mapping.

In the case where the PKINIT Request and certificate signatures are all valid but the manufacturer certificate is revoked, the KDC MAY still proceed with the MTA FQDN Request. In this case, the KDC MUST provide the revocation time in the MTA FQDN Request.

The MTA FQDN Request MUST be formatted as follows:

Table 6. MTA FQDN Request Format

Field Name	Length	Description
KRB_AP_REQ	Variable	DER-encoded, the length is in the ASN.1 header.
KRB_SAFE	Variable	DER-encoded

In the KRB_AP_REQ, only the following option is supported:

- MUTUAL-REQUIRED – mutual authentication required. This option MUST always be set.
- All other options are not supported

The encrypted authenticator in the KRB_AP_REQ MUST contain the following field, which is optional in Kerberos:

- seq-number - random value generated by the KDC

All other optional fields within the encrypted authenticator are not supported within PacketCable. In Section 6.5.2.2 there is a requirement that the recipient of a KRB_AP_REQ accepts certain negative values of seq-number; that requirement does not apply when processing the KRB_AP_REQ embedded in a received MTA FQDN message. The authenticator itself MUST be encrypted using 3-DES CBC with the Kerberos etype value des3-cbc-md5 with the session key from the ticket that is contained in this KRB_AP_REQ object. The encryption method for des3-cbc-md5 is specified in Section 6.4.2.2.

KRB_SAFE MUST contain the following field, which is optional in Kerberos:

- seq-number - same value as in the KRB_AP_REQ, to tie KRB_SAFE to KRB_AP_REQ and avoid replay attacks.

All other optional fields within KRB_SAFE are not supported within PacketCable. The keyed checksum within KRB_SAFE MUST be of type rsa-md5-des3 and MUST be computed with the session key in the accompanying KRB_AP_REQ. The method for computing an rsa-md5-des3 keyed checksum is specified in Section 6.4.3.1.

The data that is wrapped inside KRB_SAFE MUST be formatted as follows:

Table 7. KRB_SAFE Format

Field Name	Length	Description
Message Type	1 byte	1 = MTA FQDN Request
Enterprise Number	4 bytes	Network byte order, MSB first. 1 = PacketCable
Protocol Version	1 byte	2 for this version
MTA MAC Address	6 bytes	MTA MAC Address
3 MTA Pub Key Hash	20 bytes	SHA-1 hash of DER-encoded SubjectPublicKeyInfo.
Manufacturer Cert Revocation Time	4 bytes	0 = MTA Manufacturer cert not revoked Otherwise, this is UTC time, number of seconds since midnight of Jan 1, 1970, in network byte order.

Once the KDC has sent an MTA FQDN Request, it MUST save the nonce value that was contained in the seq-number field in order to validate a matching MTA FQDN Reply.

If the KDC times out before getting a reply it MUST give up and simply drop the PKINIT request with no error code returned. The KDC MUST NOT retry in this case, since it would still have to handle retries of PKINIT Request from the MTA. At the same time, after a time out the KDC SHOULD increase its time out value on the next request to the same Provisioning Server using an exponential back-off algorithm.

The Provisioning Server receiving this message MUST validate the KRB_AP_REQ and verify that it is not a replay using the procedure specified in the Kerberos standard Appendix B, also described in Section 6.5.2. After the KRB_AP_REQ has been validated, the Provisioning Server MUST also verify the KRB_SAFE component: that the checksum keyed with the session key is valid and that the seq-number field matches the KRB_AP_REQ.

If the Manufacturer Cert Revocation Time field is 0 and the Provisioning Server supports the storage of MTA public key hashes, then it MUST update the MTA public key hash in its database. If the public key hash has changed or is saved for the first time, the Provisioning Server MUST also record the time this update (to the MTA public key hash) is performed.

If the Manufacturer Cert Revocation Time field is non-zero, the Provisioning Server MUST validate that the public key hash hasn't changed from the previous update and that the revocation time is after the last update to the MTA public key hash. If not – the error code KRB_MTAMAP_ERR_PUBKEY_NOT_TRUSTED MUST be returned. If the Provisioning Server does not support storage of MTA public key hashes and the Manufacturer Cert Revocation Time field is non-zero, the same error code MUST be returned.

6.4.7.2 MTA FQDN Reply

The MTA FQDN Reply MUST be formatted as follows:

Table 8. MTA FQDN Format

Field Name	Length	Description
KRB_AP_REP	Variable	DER-encoded, the length is in the ASN.1 header.
KRB_SAFE	Variable	DER-encoded

The encrypted part of the KRB_AP_REP MUST contain the following field, which is optional in Kerberos:

- seq-number - echoes the value in the KRB_AP_REQ

All other optional fields within the encrypted part of the KRB_AP_REP are not supported within PacketCable. It MUST be encrypted using 3-DES CBC with the Kerberos etype value des3-cbc-md5 and MUST be computed with the session key from the preceding KRB_AP_REQ. The encryption method for des3-cbc-md5 is specified in Section 6.4.2.2.

KRB_SAFE MUST contain the following field, which is optional in Kerberos:

- seq-number - same value as in the KRB_AP_REP, to tie KRB_SAFE to KRB_AP_REP and avoid replay attacks.

All other optional fields within KRB_SAFE are not supported within PacketCable. The keyed checksum within KRB_SAFE MUST be of type rsa-md5-des3 and MUST be computed with the session key from the preceding KRB_AP_REQ. The method for computing an rsa-md5-des3 keyed checksum is specified in Section 6.4.3.1.

The data that is wrapped inside KRB_SAFE MUST be formatted as follows:

Table 9. KRB_SAFE Data Format

Field Name	Length	Description
Message Type	1 byte	2 = MTA FQDN Reply
Enterprise Number	4 bytes	Network byte order, MSB first. 1 = PacketCable
Protocol Version	1 byte	2 for this version
MTA FQDN	variable	MTA FQDN
MTA IP Address	4 bytes	MTA-IP Address (MSB first)

After the KDC receives this reply message, it MUST validate the integrity of both the KRB_AP_REP and KRB_SAFE objects (see Appendix B) and MUST also verify that the value of the seq-number field is the same for both. If this integrity check fails, the KDC MUST immediately discard the reply and proceed as if the message had never been received (e.g., if the KDC was waiting for a valid MTA FQDN Reply it should continue to do so).

The Provisioning Server MAY set the MTA IP Address field of the MTA FQDN Reply to zero. If the KDC receives an MTA FQDN REPLY with a non-zero MTA IP Address field, it MUST compare it to the IP address contained in the AS Request. If this check fails, then the KDC MUST NOT respond to the AS Request.

6.4.7.3 MTA FQDN Error

If the Provisioning Server is able to successfully parse the KRB_AP_REQ and the ticket that is inside of it, but the MTA FQDN Request is rejected, it MUST return an error message.

All errors MUST be returned as a KRB_ERROR message, as specified in Appendix B. It MUST include typed-data of REQ-SEQ to bind the error message to the sequence number from the authenticator in the KRB_AP_REQ. Also, the error message MUST include the optional e-cksum member, which is the keyed hash over the KRB_ERROR message. The checksum type MUST be rsa-md5-des3 and MUST be computed with the session key from the preceding KRB_AP_REQ, as specified in Section 6.4.3.1. In the case that the client time field inside KRB_AP_REQ differs from the Provisioning Server's clock by more than the maximum allowable clock skew, a clock skew error MUST be handled as specified in Section 6.5.2.3.2.

If the error is application-specific (not a Kerberos-related error), then KRB_ERROR MUST include typed-data of type TD-APP-DEFINED-ERROR (value 106). The value of this typed-data is specified in Appendix B as follows:

```
AppSpecificTypedData ::= SEQUENCE {
  oid        [0] OPTIONAL OBJECT IDENTIFIER,
                 -- identifies the application
```

```
  data-value [1] OCTET STRING
                 -- application specific data
}
```

Inside AppSpecificTypedData the oid field MUST be set to:

enterprises (1.3.6.1.4.1) cableLabs (4491) clabProjects (2) clabProjPacketCable (2) pktcSecurity (4) errorCodes (1) FQDN (3)

The data-value field MUST correspond to the following typed-data value:

```
PktcKrbMtaMappingError ::= SEQUENCE {
  e-code [0] INTEGER,
  e-text [1] GeneralString OPTIONAL,
  e-data [2] OCTET STRING OPTIONAL
}
```

The e-code field MUST correspond to one of the following error code values:

KRB_MTAMAP_ERR_NOT_FOUND	1	MTA MAC Address not found
KRB_MTAMAP_ERR_PUBKEY_NOT_TRUSTED	2	MTA public key is not trusted
KRB_MTAMAP_VERSION_UNSUP	3	Unsupported Version Number
KRB_MTAMAP_MSGTYPE_UNKNOWN	4	Unrecognized Message Type
KRB_MTAMAP_ENTERPRISE_UNKNOWN	5	Unrecognized Enterprise Number
KRB_MTAMAP_NOT_YET_VALID	6	MTA not yet valid
KRB_MTAMAP_ERR_GENERIC	7	Generic MTA name mapping error

The optional e-text field can be used for informational purposes (i.e., logging, network troubleshooting) and the optional e-data field is reserved for future use to transport any application data associated with a specific error.

Upon receipt of a KRB_ERROR from the Provisioning Server, the KDC MUST check the validity of the checksum. If the KRB_ERROR passes the validity check, the KDC MUST send a corresponding KRB_ERROR to the MTA (as specified in 6.4.2.1.2), in response to the PKINIT Request. The application specific MAC-FQDN error codes MUST be mapped to Kerberos error codes in the error reply to the MTA according to Table 10.

Table 10. Mapping of KRB_MTAMAP_ERR to KRB_ERR

KRB_MTAMAP_ERR_NOT_FOUND	KDC_ERR_C_PRINCIPAL_UNKNOWN
KRB_MTAMAP_ERR_PUBKEY_NOT_TRUSTED	KDC_ERR_CLIENT_REVOKED
KRB_MTAMAP_VERSION_UNSUP	KRB_ERR_GENERIC
KRB_MTAMAP_MSGTYPE_UNKNOWN	KRB_ERR_GENERIC
KRB_MTAMAP_ENTERPRISE_UNKNOWN	KRB_ERR_GENERIC
KRB_MTAMAP_NOT_YET_VALID	KDC_ERR_CLIENT_NOTYET
KRB_MTAMAP_ERR_GENERIC	KRB_ERR_GENERIC

6.4.8 Server Key Management Time Out Procedure

The Kerberos client MUST implement a retransmission strategy using exponential back-off with configurable initial and maximum retransmission timer values for any KDC or application server requests that have not been acknowledged by the server. The Kerberos client MUST update the client timestamp field with the current time-of-day reading for each such retry. During an exponential back-off, when a previous time out value was T_i, then the next time out value, value T_{i+1}, MUST satisfy the following criteria:

$1.5 * T_i <= T_{i+1} <= 2.5 * T_i$

After successfully processing an AS Request or TGS Request and generating a corresponding reply, the KDC MUST save:

- The AS Request or TGS Request (e.g., the full AS Request / TGS Request or a hash of the AS Request / TGS Request)
- The full KDC reply

The KDC MUST maintain this information for all requests with the client time field that is within the time window ($T - \Delta T_{MAX}$, $T + \Delta T_{MAX}$), where T is the current time and ΔT_{MAX} is the maximum clock skew that is allowed by KDC policy.

The KDC MAY also save:

- The client principal identifier
- The information that uniquely identifies the client pre-authentication field in the AS Request (PKINIT or encrypted timestamp in the case of non public key AS Request) or TGS Request (PA-TGS-REQ).

The KDC MAY maintain this information for all requests with the client time field that is within the time window ($T - \Delta T_{MAX}$, $T + \Delta T_{MAX}$), where T is the current time and ΔT_{MAX} is the maximum clock skew that is allowed by KDC policy. If the AS Request or TGS Request is identical to the one previously received, the KDC MUST respond with the same reply message. If only the principal name and pre-authenticator (PKINIT, encrypted timestamp or PA-TGS-REQ) match, then the KDC MUST perform one of the following:

- If the received AS Request or TGS Request passes all other error checks, the KDC may reply with a cached reply message
- Reject this message as a replay

The MTA may have learned several IP addresses for a KDC or application server (refer to Section 6.4.5.2 for more information on obtaining IP addresses from Realm Names and forming a local list of IP addresses based on prioritization). If the number of retransmissions for a KDC IP address has reached its maximum configured value and there are more IP addresses for the same KDC that have not been tried, then the MTA MUST direct the retransmissions to the remaining alternate addresses in its local list. Each time that the MTA switches to a new KDC IP address for retransmissions, it MUST start a new exponential back-off procedure. If there are no more KDC IP addresses to try, then the MTA SHOULD actively query the name server in order to detect the possible change of KDC network interfaces, regardless of the Time To Live (TTL) associated with the DNS record to see if any other IP addresses have become available. If there are new IP Addresses discovered, the MTA MUST go through the retransmission strategy again for the newly discovered IP Addresses.

For Kerberized key management with application servers, when an application layer is informed that key management with a particular IP address failed, it is normally up to the application layer to select the next IP address. The switch over algorithm between multiple IP addresses mapped to the same FQDN is specified by each corresponding application protocol. For example, in the case of the Kerberized key management between the MTA and the CMS, refer to the NCS specification [2]. There are also cases when key management is performed independent of the application layer, e.g., to pre-establish security associations during MTA initialization. In those cases, it is up to a specific MTA implementation to decide if to fail over and how to fail over to another application server IP address.

An application server may not respond to application messages (e.g., NCS messages) from the MTA. This may occur if the MTA has valid security parameters with the application server, but the security parameters on the server have been lost or corrupted (e.g., the CMS rebooted and lost all IPsec Security Associations).

In the case of NCS signaling, an MTA MUST no longer use any previously established IPsec SAs with a particular CMS each time the NCS backoff and retry algorithm places an MTA endpoint controlled by that CMS into a DISCONNECTED state. After an MTA endpoint has moved to a DISCONNECT state, it will

start sending RSIP/disconnect NCS messages which will need to be protected by newly established IPsec SAs.

6.4.9 Service Key Versioning

The service key that is shared between a KDC and an application server, to encrypt/decrypt service tickets, is a versioned key (refer to Appendix B). This key may be changed either due to a routine key refresh, or because it was compromised. When the Service key is changed, the application server MUST retain the older key for a period of time that is at least as long as the ticket lifetime used when issuing service tickets (i.e., up to 7 days). In the case of a routine service key change, the application server MUST accept any ticket that is encrypted with an older key that it has retained and is still valid (not compromised). This key versioning on the application server will prevent against many MTAs from suddenly flooding a KDC with PKINIT Requests for new tickets.

If a service key is changed because it has been compromised, the application server MUST flag all older key versions it has retained as invalid and reject any AP Request that contains a ticket that is encrypted with one of these invalid keys. When rejecting the AP Request, the application server MUST respond as specified in Appendix B with a KRB_AP_ERR_BADKEYVER error. The application server MUST still decrypt the rejected ticket, using the invalid service key, in order to extract the session key. This session key is needed to securely bind the KRB_ERROR reply message to the AP Request message using a keyed checksum (see Section 6.5.2.3.1). Note that this step is necessary in order to prevent denial-of-service attacks, which could otherwise occur if the MTA was unable to verify the authenticity of the KRB_ERROR message.

Upon receiving this error reply, the MTA MUST discard the service ticket which is no longer valid and fetch a new one from its KDC.

6.5 Kerberized Key Management

6.5.1 Overview

This section specifies how Kerberos tickets are used to perform key management between a client and an Application Server, where a client is able to get a Kerberos ticket for the server but not the other way around.

The same protocol described here applies in a symmetric case – where both sides of a key management interface are able to get a ticket for each other, i.e., each side is both a client and a server. In the symmetric case only the AP Request and AP Reply messages apply.

The Kerberos session key is used in the AP Request and AP Reply messages that are exchanged in order to re-establish security parameters. Subkeys from the AP-REQ and AP-REP are used to derive all of the secret keys used for both directions. The AP Request and AP Reply messages are small enough to fit into a standard UDP packet, not requiring fragmentation.

A Kerberos AP Request / Reply exchange MAY occur periodically, to insure that there are always valid security parameters between the client and the Application Server. It MAY also occur on-demand, where the security parameters are allowed to time out and are re-established the next time that application traffic needs to be sent over a secure link.

The UDP port used for all key management messages between the client and the Application Server MUST be 1293 (on both devices).

A recipient of any Kerberized Key Management message that doesn't fully comply with the PacketCable requirements MUST reject the message.

6.5.2 Kerberized Key Management Messages

The following figure illustrates an AP Request / AP Reply exchange:

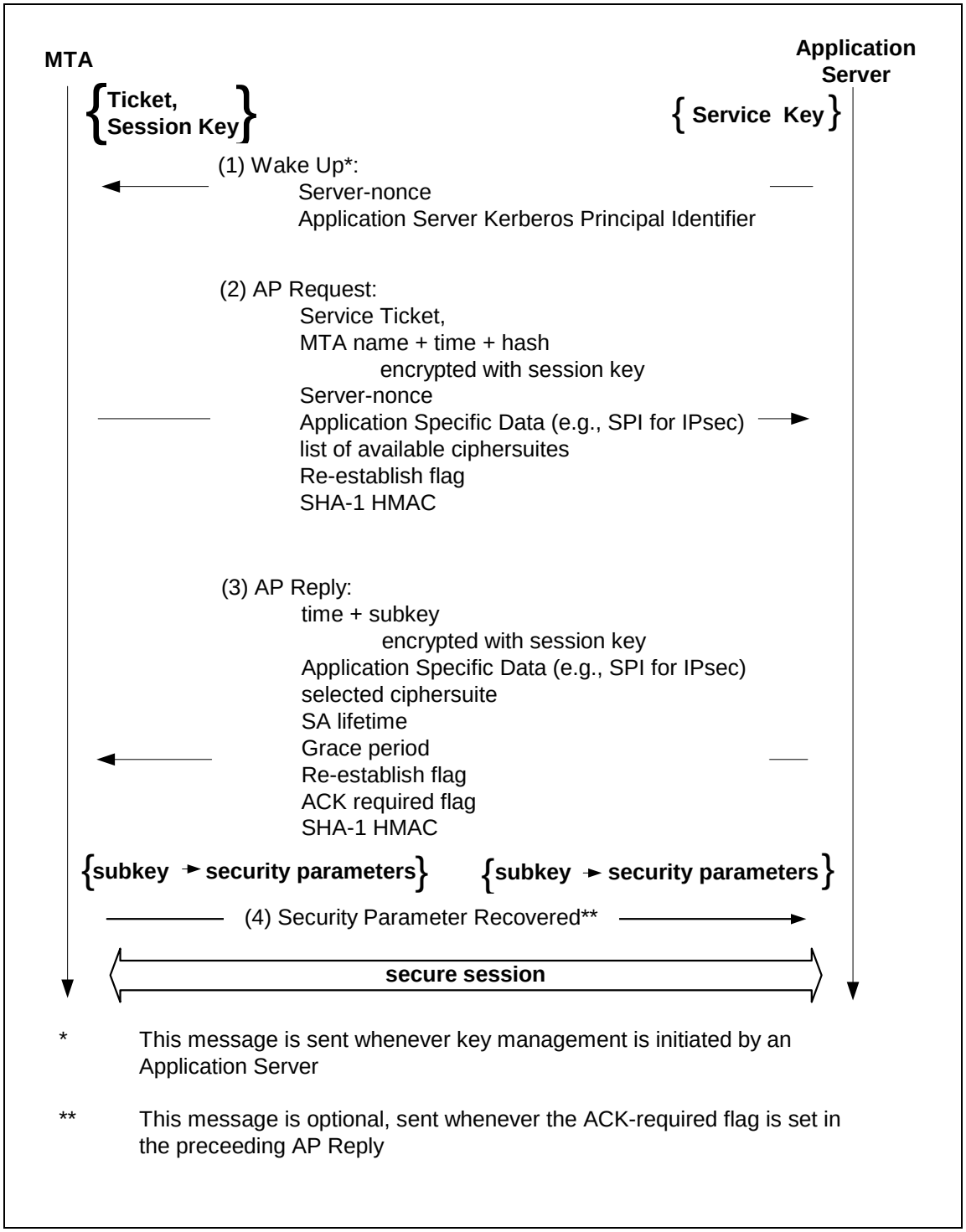


Figure 8. Kerberos AP Request / AP Reply Exchange

(1) Wake Up - An Application Server sends this message when it initiates a new key management exchange.

To prevent denial-of-service attacks, this message includes a Server-nonce field – a random value generated by the Application Server. The Client includes the exact value of this Server-nonce in the subsequent AP Request.

This message also contains the Server Kerberos Principal Identifier, used by the Client to find or to obtain a correct Kerberos ticket for that Application Server.

The Wake Up message MUST be formatted as the concatenation of the following fields:

- Key Management Message ID – 1 byte value. Always set to 0x01.
- Domain of Interpretation (DOI) – 1 byte value. Specifies the target protocol for which security parameters are established.

DOMAIN OF INTERPRETATION VALUES	
VALUE	TARGET PROTOCOL
1	IPsec
2	SNMPv3

- Protocol Version – 1 byte. The high order nibble is the major version number, and the lower order nibble is the minor version number. For PacketCable, the major number MUST be 1, and the minor number MUST be 0.
- Server-nonce – a 4-byte random binary string. Its value MUST NOT be all 0's.
- Server Kerberos Principal Identifier – a printable, null-terminated ASCII string, representing the Kerberos Principal Identifier of the Application Server, as defined in Section 6.4.5.

Once the Application Server has sent a Wake Up, it MUST save the Server-nonce. The Application Server MUST keep this nonce in order to validate a matching AP Request. In the case of a time out, the Application Server MUST adhere to the exponential retry backoff procedure described in Section 6.4.8. The Application Server MUST begin each retry by re-sending a Wake Up message with a new server-nonce value. When the "Timeout Procedure" has completed without success, the Application Server MUST discard the server-nonce from the last retry, after which it will no longer accept a matching AP Request.

(2) AP Request – MUST be sent by the Client in order to establish a new set of security parameters. Any time that the Client receives a Wake Up message from a valid application server that is listed as part of client configuration data, it MUST respond with the AP Request message specified below. If a client receives a Wake Up message from an unknown application server, the client MUST NOT respond.

In addition, this document specifies the use of this message by the Client to periodically establish a new set of security parameters with the Application Server – see Section 6.5.4.2. It also specifies the use of this message by the Client to establish a new set of security parameters with the Application Server, when the Client somehow loses the security parameters (e.g., after a reboot) – see Section 6.5.3.5.

The Client starts out with a valid Kerberos ticket, previously obtained during a PKINIT exchange. The Application Server starts out with its Service Key that it can use to decrypt and validate Kerberos tickets.

The Client sends an AP Request that includes a ticket and an authenticator, encrypted with the session key. The Application Server gets the session key out of the ticket and uses it to decrypt and then validate the authenticator.

The AP Request includes the Kerberos KRB_AP_REQ message along with some additional information, specific to PacketCable. It MUST consist of the concatenation of the following fields:

- Key Management Message ID – 1 byte value. Always set to 0x02.
- Domain of Interpretation (DOI) – 1 byte value. Specifies the target protocol for which security parameters are established. See Table above.
- Protocol Version – 1 byte. The high order nibble is the major version number, and the lower order nibble is the minor version number. For PacketCable, the major number MUST be 1, and the minor number MUST be 0.
- KRB_AP_REQ – DER encoding of the KRB_AP_REQ Kerberos message, as specified in Appendix B.
- Server-nonce – a 4-byte random binary string. If this AP Request is in response to a Wake Up, then the value MUST be identical to that of the Server-nonce field in the Wake Up message. If this AP Request is in response to a Rekey, next Section 6.5.2.1, then the value MUST be identical to that of the Server-nonce field in the Rekey message. Otherwise, the value MUST be all 0's.
- Application-Specific Data – additional information that must be communicated by the client to the server, dependent on the target protocol for which security is being established (e.g., IPsec or SNMPv3).
- List of ciphersuites available at the Client:

  Number of entries in this list (1 byte)

  Each entry has the following format:

Authentication Algorithm (1 byte)	Encryption Transform ID (1 byte)

  The actual values of the authentication algorithms and encryption transform Ids are dependent on the target protocol.
- Re-establish flag – a 1-byte Boolean value. When the value is TRUE (1), the Client is making an attempt to automatically establish a new set of Security Parameters before the old ones expire. Otherwise the value is FALSE (0).
- SHA-1 HMAC - (20 bytes) over the contents of this message, not including this field. The 20-byte key for this HMAC is determined by taking a SHA-1 hash of the session key.

- Whenever the AP Request is received (by the Application Server), it MUST verify the value of this HMAC. If this integrity check fails, the Application Server MUST immediately discard the AP Request and proceed as if the message had never been received (e.g., if the Application Server was waiting for a valid AP Request it should continue to do so).

Once the client has sent an AP Request, it MUST save the nonce value that was contained in the seq-number field (a different nonce from the server-nonce specified above) along with the Server Kerberos Principal Identifier in order to validate a matching AP Reply. If the client generated this AP Request on its own, it MUST adhere to the exponential retry backoff procedure described in Section 6.4.8.

If the AP Request was generated in response to a message sent by the Application Server (Wake Up or Rekey), then the client MUST save the nonce and Server Kerberos Principal Identifier until the time specified by the appropriate Key Management MIB variables (pktcMtaDevProvSolicitedKeyTimeout for Prov Server, pktcMtaDevCmsSolicitedKeyTimeout for CMS).

After the timeout has been exceeded or when the "Timeout Procedure" has completed without success, the client MUST discard this (nonce, Server Kerberos Principal Identifier) pair, after which it will no longer accept a matching AP Reply.

If the MTA generated an AP Request on its own and has reached the maximum number of retries with a particular application server IP address failing to get an AP Reply, it must retry with alternate application server IP addresses as specified in Section 6.4.8.

In the case that the Server-nonce is 0 (not filled in) and the Application Server is currently waiting for a reply to a Wake Up or Rekey message from a client at this IP address, it MUST reject the AP Request and not reply to the client. If the Application Server is not waiting for a reply to a Wake Up or Rekey message, it MUST verify that this AP Request is not a replay using the procedure specified in the Kerberos standard (Appendix B):

- If the timestamp in the AP Request differs from the current Application Server time by more than the acceptable clock skew then Application Server MUST reply with an error message specified in Section 6.5.2.3.2.
- If the realm, Application Server name, along with the Client name, time and microsecond fields from the Kerberos Authenticator (in the AP Request) match any recently-seen such tuples, the KRB_AP_ERR_REPEAT error MAY be returned.
- The Application Server MUST remember any authenticator presented within the acceptable clock skew, so that a replay attempt is guaranteed to fail.
- If the Application Server loses track of any authenticator presented within the acceptable clock skew, it MUST reject all requests until the interval has passed.

In the case that the Server-nonce is not 0, the Application Server MAY follow the above procedure in order to fully conform with the Kerberos specification (Appendix B). In this case, the above procedure is not required because matching the Server-nonce in the Wake Up or Rekey message against the Server-nonce in the AP Request also prevents replays.

(3) AP Reply – Sent by the Application Server in response to AP Request.
The AP Reply MUST include a randomly generated subkey (inside the Kerberos KRB_AP_REP message), encrypted with the same session key.

The AP Reply includes the Kerberos KRB_AP_REP message along with some additional information, specific to PacketCable. It MUST consist of the concatenation of the following fields:

- Key Management Message ID – 1 byte value. Always set to 0x03.
- Domain of Interpretation (DOI) – 1 byte value. Specifies the target protocol for which security parameters are established. See Table in Section 6.5.2.
- Protocol Version – 1 byte. The high order nibble is the major version number, and the lower order nibble is the minor version number. For PacketCable, the major number MUST be 1, and the minor number MUST be 0.
- KRB_AP_REP – DER encoding of the KRB_AP_REP Kerberos message, as specified in Appendix B.
- Application-Specific Data – additional information that must be communicated by the server to the client, dependent on the target protocol for which security is being established (e.g., IPsec or SNMPv3).
- Selected ciphersuite for the target protocol, using the same format as defined for AP Request. The number of entries in the list MUST be one.
- Security parameters lifetime – a 4-byte value, MSB first, indicating the number of seconds from now, when these security parameters are due to expire.
- Grace period – a 4-byte value in seconds, MSB first. This indicates to the client to start creating a new set of security parameters (with a new AP Request / AP Reply exchange) when the timer gets to within this period of their expiration time.

- Re-establish flag – a 1-byte Boolean value. When the value is TRUE (1), a new set of security parameters MUST be established before the old one expires. When the value is FALSE (0), the old set of security parameters MUST be allowed to expire.
- ACK-required flag – a 1-byte Boolean value. When the value is TRUE (1), the AP Reply message requires an acknowledgement, in the form of the Security Parameter Recovered message
- SHA-1 HMAC – (20 bytes) over the contents of this message, not including this field. The 20-byte key for this HMAC is determined by taking a SHA-1 hash of the session key.

Whenever the AP Reply is received (by the Client) it MUST:

- verify the value of HMAC field in AP Reply. If HMAC integrity check fails, the Client MUST immediately discard the AP Reply.
- verify that the AP Reply Source IP Address matches the AP Request Destination IP Address in the list of outstanding AP Requests. The Client MUST immediately discard the AP Reply, which cannot be matched for the corresponding AP Request.
- verify that the nonce value contained in the seq-number field in AP Reply matches the one in the corresponding AP Request. The Client MUST immediately discard the AP Reply if seq-number field value in AP Reply does not match.

If the AP Reply is discarded, the Client MUST proceed as if the message had never been received (e.g. if the Client was waiting for a valid AP Reply it should continue to do so).

Once the Application Server has sent an AP Reply with the ACK-required flag set, it MUST compute the expected value in the Security Parameter Recovered message and save it for an appropriate timeout period during which it will accept a matching Security Parameter Recovered Message. Once the appropriate timeout period is exceeded, the Application Server MUST discard the saved values and no longer accept a matching Security Parameter Recovered Message.

Each time the Application Server times out waiting for the Security Parameter Recovered message, it MUST continue with the exponential back-off algorithm until all retries have been exhausted, as specified in Section 6.4.8. The Application Server MUST begin each retry by re-sending a Wake Up message with a new server-nonce value.

(4) Security Parameter Recovered – Sent by the Client to the Application Server to acknowledge that it received an AP Reply and successfully set up new Security Parameters. This message is only sent when ACK-required flag is set in the AP Reply.

This message MUST consist of the concatenation of the following:

- Key Management Message ID – 1 byte value. Always set to 0x04.
- Domain of Interpretation (DOI) – 1 byte value. Specifies the target protocol for which security parameters are established. See Table in Section 6.5.2.
- Protocol Version – 1 byte. The high order nibble is the major version number, and the lower order nibble is the minor version number. For PacketCable, the major number MUST be 1, and the minor number MUST be 0.
- HMAC – a 20-byte SHA-1 HMAC of the preceding AP Reply message. The 20-byte key for this HMAC is determined by taking a SHA-1 hash of the subkey from the AP Reply.

If the receiver (Application Server) gets a bad Security Parameter Recovered message that does not match an AP Reply, the Application Server MUST discard it and proceed as if this Security Parameter Recovered message was never received.

6.5.2.1 Rekey Messages

The Rekey message replaces the Wake Up message and provides better performance, whenever a receiver (Application Server) wants to trigger the establishment of a Security Parameter with a specified Client. The

Rekey message requires the availability of the shared Server Authentication Key, which is not always available. Thus, support for the Wake Up message is still required.

The Rekey message was added specifically for use with the NCS-based clustered Call Agents, potentially consisting of multiple IP addresses and multiple hosts. Any IP address or host within one cluster needs the ability to quickly establish a new Security Parameter with a Client, without a significant impact to the ongoing voice communication.

The use of the Rekey message eliminates the need for the AP Reply message, thus reducing the key management overhead to a single roundtrip. This is illustrated in the following diagram:

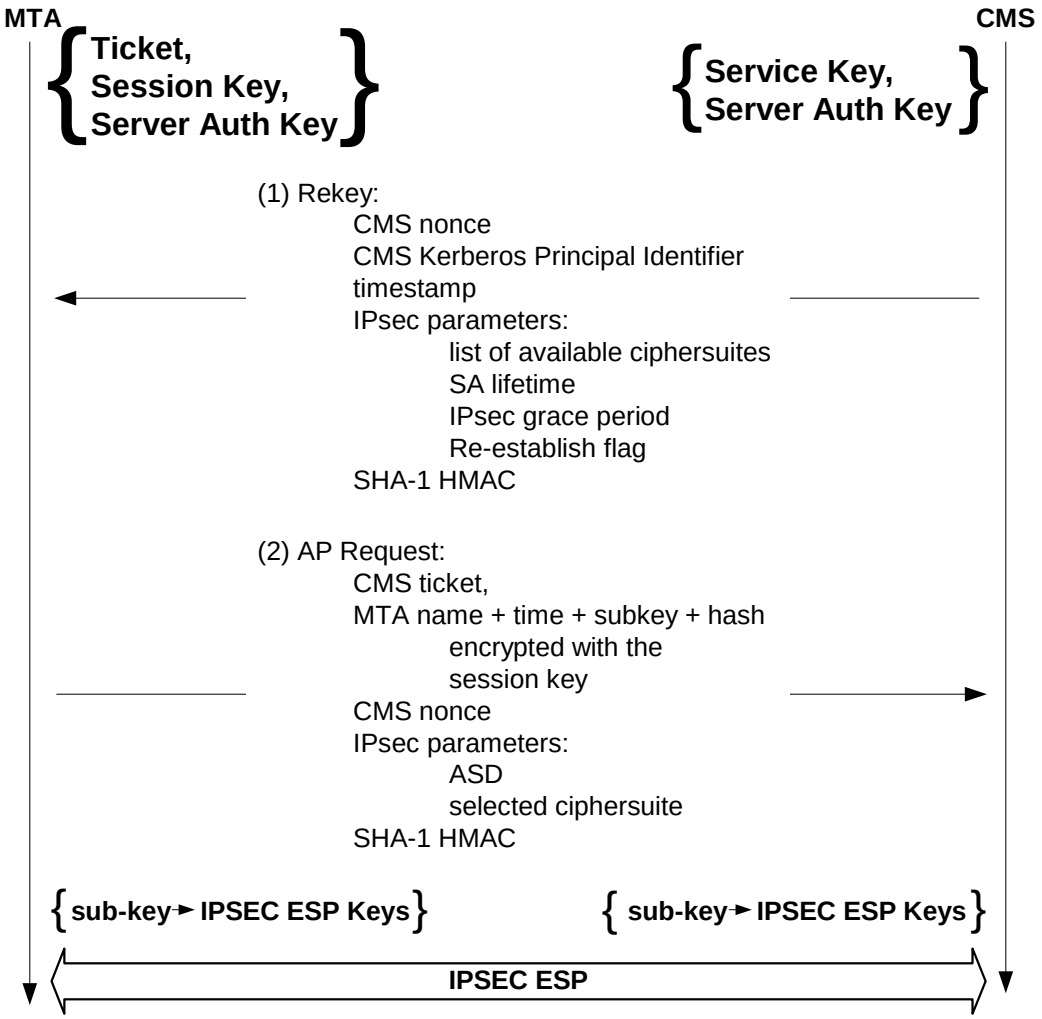


Figure 9. Rekey Message to Establish a Security Parameter

The messages listed in this diagram are defined as follows:

(1) Rekey – sent by the Application Server to establish a new set of Security Parameters. It MUST be a concatenation of the following:

- Key Management Message ID – 1 byte value. Always set to 0x05.
- Domain of Interpretation (DOI) – 1 byte value. Specifies the target protocol for which security parameters are established. See Table in Section 6.5.2.

- Protocol Version – 1 byte. The high order nibble is the major version number, and the lower order nibble is the minor version number. For PacketCable, the major number MUST be 1, and the minor number MUST be 0.
- Server-nonce – a 4-byte random binary string. Its value MUST NOT be all 0's.
- Server Kerberos Principal Identifier – a printable, null-terminated ASCII string, representing the Kerberos Principal Identifier of the Application Server, as defined in Section 6.4.5. This allows the Client to both find the right Server Authentication Key and to pick the right Kerberos ticket for the subsequent AP Request message.
- Timestamp – a string of the format YYMMDDhhmmssZ, representing UTC time. This string is not NULL-terminated.
- Application-Specific Data – additional information that must be communicated by the server to the client, dependent on the target protocol for which security is being established (e.g., IPsec).
- List of ciphersuites available at the server – see above specification for the AP Request message.
- Security parameters lifetime – a 4-byte value, MSB first. This indicates the number of seconds from now, when this set of security parameters is due to expire.
- Grace period – a 4-byte value in seconds, MSB first. This indicates to the client to start creating a new set of security parameters (with a new AP Request / AP Reply exchange) when the timer gets to within this period of their expiration time.
- Re-establish flag – a 1-byte Boolean value. When the value is TRUE (1), a new set of security parameters MUST be established before the old one expires. When the value is FALSE (0), the old set of security parameters MUST be allowed to expire.
- SHA-1 HMAC – over the concatenation of all of the above listed fields.

The Server Authentication Key used for this HMAC is uniquely identified by the following name pair (client principal name, server principal name). This key MUST be updated at the Application Server right after it sends an AP Reply message. It MUST be set to a (20-byte) SHA-1 hash of the Kerberos session key used in that AP Reply. The Client MUST also update this key as soon as it receives the AP Reply. (Note that multiple AP Replies will continue using the same Kerberos session key, until it expires. That means that the derived Server Authentication Key may have the same value as the old one.)

It is possible, that the Application Server sends a Rekey message as soon as it sends an AP Reply (from another IP address), and before the Client is able to derive the new Server Authentication Key. In that case, the Client will not authenticate the Rekey message and the Application Server will have to retry.

Similarly, after sending an AP Reply the Application Server might immediately send an IP packet using the just established Security Parameter, when the Client is not yet ready to receive it. In this case, the Client will reject the packet and the Application Server will have to retransmit.

Both of these error cases could be completely avoided with a 3-way handshake (a Client acknowledging an AP Reply with a Security Parameter Recovered message).

Whenever the Rekey message is received (by the Client), it MUST verify the value of this HMAC. If this integrity check fails, the Client MUST immediately discard this message and proceed as if the message had never been received.

Once the Application Server has sent a Rekey, it MUST save the server-nonce in order to validate a matching AP Request. In the case of a time out, the Application Server MUST adhere to the exponential retry backoff procedure described in Section 6.4.8. The Application Server MUST begin each retry by re-sending a Rekey message with a new server-nonce value. When the "Timeout Procedure" has completed without success, the Application Server MUST discard the server-nonce from the last retry, after which it will no longer accept a matching AP Request.

When this Rekey message is received and validated by the Client, all previously existing outgoing Security Parameters with this Application Server IP address MUST be removed at this time. If the Client previously had a timer set for automatic refresh of Security Parameters with this Application Server IP address, that automatic refresh MUST be reset or disabled.

The Client MUST verify that this Rekey message is not a replay using the procedure similar to the one for AP Request in the Kerberos standard Appendix B:

- If $|T_{CMS} - (T_{MTA} + Skew)| >$ The acceptable Clock Skew then the Client MUST drop the message. Here, T_{CMS} is the timestamp in the Rekey message and T_{MTA} is the reading of the MTA local clock. Skew is the saved difference between the Application Server and MTA clock. PktcSrvrToMtaMaxClockSkew is currently in the MTA MIB (see [25] and [48]) as the variable pktcMtaDevCmsMaxClockSkew.
- If the Server-nonce, principal name and timestamp fields match any recently seen (within the pktcSrvrToMtaMaxClockSkew) Rekey messages, then the Client MUST drop the message.

(2) AP Request – MUST be sent by the Client as a response to a Rekey message. Unlike the AP Request message described above, this one MUST also include the subkey (inside KRB_AP_REQ ASN.1 structure). KRB_AP_REQ will have a Kerberos flag set, indicating that an AP Reply MUST NOT follow.

The format of the AP Request is as specified above in Section 6.5.2. The only difference is that the list of ciphersuites here MUST contain exactly one entry – the ciphersuite selected by the client from the list provided in the Rekey message.

Right before the client sends out this AP Request, it MUST establish the security parameters with the corresponding server IP address. If the corresponding Rekey message had the Re-establish flag set, the client MUST be prepared to automatically re-establish new security parameters, as specified in Section 6.5.

Once this AP Request is received and verified by the Application Server, the server MUST also establish the security parameters.

6.5.2.2 PacketCable Profile for KRB_AP_REQ / KRB_AP_REP Messages

In the KRB_AP_REQ, only the following option is supported:

- MUTUAL-REQUIRED – mutual authentication required. When this option is set, the server MUST respond with an AP Reply message. When this option is not set, the AP Reply message MUST NOT be sent in reply.

  All other options MUST NOT be set. If an application server receives a request containing the unsupported option USE-SESSION-KEY, it MUST return an error with the error code KRB_AP_ERR_METHOD. If an application server receives a request containing any other unsupported options, it MUST return an error with the error code KRB_ERR_GENERIC.

  When MUTUAL-REQUIRED is set, the encrypted authenticator in the KRB_AP_REQ MUST contain the following field, which is optional in Kerberos:

  - seq-number MUST contain a pseudo-random number generated by the client (to be used as a nonce).
  - The server MUST accept otherwise-valid KRB-AP-REQ messages that contain a seq-number in the range –2^31 to –1.

When MUTUAL-REQUIRED is not set, the encrypted authenticator MUST contain the following field that is optional in Kerberos.

- subkey – used to generate security parameters for the target protocol. The subkey type MUST be set to –1. The actual subkey length is dependent on the target protocol.

When MUTUAL-REQUIRED is set, the target protocol is IPsec and the client is an MTA, the client MAY include the subkey field; in the case that the target protocol is IPsec and the client is other than an MTA,

the client SHOULD include the subkey field. For IPsec, the subkey, if present, MUST contain a pseudo-random number of length 46 octets generated by the client.

Other optional fields in the authenticator MUST NOT be present. If the authenticator contains the authorization-data field, the application server MUST return an error with the error code KRB_ERR_GENERIC. If the authenticator contains any other optional fields (apart from subkey and authorization-data), the application server MUST ignore those fields.

The negative key type is used to indicate that it is application-specific and not defined in the Kerberos specification. When the Kerberos specification is updated to include this key type, the PacketCable spec will be updated accordingly.

The authenticator itself MUST be encrypted using 3-DES CBC with the Kerberos etype value des3-cbc-md5 as it is specified in Section 6.4.2.2.

In the encrypted part of the KRB_AP_REP, the optional subkey field MUST be used for PacketCable. Its type and format MUST be the same as when it appears in the KRB_AP_REQ (see above).

The optional seq-number MUST be present, and MUST echo the value that was sent by the client in the KRB_AP_REQ. In this context, the seq-number field is used as a random nonce. The encrypted part of the KRB_AP_REP MUST be encrypted with the Kerberos etype value des3-cbc-md5 as specified in Section 6.4.2.2.

6.5.2.3 Error Handling

6.5.2.3.1 Error Reply

If the Application Server is able to successfully parse the AP Request and the ticket that is inside of it, but the AP Request is rejected, it MUST return an error message. This error message MUST be formatted as the concatenation of the following fields:

- Key Management Message ID – 1 byte value. Always set to 0x06.
- Domain of Interpretation (DOI) – 1 byte value. Specifies the target protocol for which security parameters are established. See Section 6.5.2.
- Protocol Version – 1 byte value. The high order nibble is the major version number and the lower order nibble is the minor version number. For PacketCable the major version number MUST be 1 and the minor version number MUST be 0.
- KRB_ERROR – Kerberos error message as specified in Appendix B. It MUST include typed-data of REQ-SEQ to bind the error message to the sequence number from the authenticator in the AP-REQ message. The value encapsulated by the REQ-SEQ typed data MUST be the same as the value of the seq-number that was sent by the client in the KRB_AP_REQ. Also, the error message MUST include the optional e-cksum member, which is the keyed hash over the KRB_ERROR message. The checksum type MUST be rsa-md5-des3, as it is specified in Section 6.4.3.1.

If the error is application-specific (not a Kerberos-related error), then the KRB_ERROR MUST include typed-data of type TD-APP-DEFINED-ERROR (value 106). The value of this typed-data is the following ASN.1 encoding (specified in Appendix B):

```
AppSpecificTypedData ::= SEQUENCE {
  oid        [0] OPTIONAL OBJECT IDENTIFIER,
                 -- identifies the application
  data-value [1] OCTET STRING
                 -- application specific data
}
```

Both the oid and the data-value fields inside AppSpecificTypedData are specified separately for each DOI.

Upon receiving this error reply, the Client MUST verify both the keyed checksum and the REQ-SEQ field, to make sure that it matches the seq-number field from the authenticator in the AP Request.

If the Application Server is not able to successfully parse the AP Request and the ticket, it MUST drop the request and it MUST NOT return any response to the Client. In case of a line error, the Client will time out and re-send its AP Request. If the verification has failed, then the MTA MUST ignore this error message and continue waiting for the reply as if the error message was never received.

When a client receives an error message, in some cases this specification calls for the client to take some recovery steps and then send a new AP Request message. When a client is responding to an error message, it is not a retry and MUST NOT be considered to be part of the client's back-off and retry procedure specified in Section 6.4.8. The client MUST reset its timers accordingly, to reflect that the AP Request in response to an error message is not a retry.

Although this specification calls for an application server to return some specific error codes under certain error conditions, in the case when a server is repeatedly getting the same error from the same client IP address, it MAY at some point choose to stop sending back any further replies (errors or otherwise) to this client.

6.5.2.3.2 Clock Skew Error

When the Application Server clock and the client clock are off by more than the limit for a clock skew, an error code KRB_AP_ERR_SKEW MUST be returned. The value for the maximum clock skew allowed by the Application Server MUST NOT exceed 5 minutes. The optional client's time in the KRB_ERROR MUST be filled out, and the client MUST compute the difference (in seconds) between the two clocks based upon the client and server time contained in the KRB_ERROR message. The client SHOULD store this clock difference in non-volatile memory and MUST use it to adjust Kerberos timestamps in subsequent AP Request messages by adding the clock skew to its local clock value each time. The client MUST maintain a separate clock skew value for each realm and MAY share the same clock skew between the KDC and various application servers within that realm. The clock skew values are intended for uses only within the Kerberos protocol and SHOULD NOT otherwise affect the value of the local clock (since a clock skew is likely to vary from realm to realm).

In the case that an AP Request failed due to a clock skew error, a client MUST immediately retry after adjusting the Kerberos timestamp inside the AP Request message.

Additionally, the Client MUST validate the time offset returned in the clock skew error, to make sure that it does not exceed a maximum allowable amount. This maximum time offset MUST not exceed 1 hour. This Client check against a maximum time offset protects against an attack, where a rogue KDC attempts to fool a Client into accepting an expired KDC certificate (later, during the next PKINIT exchange).

6.5.2.3.3 Handling Ticket Errors After a Wake Up

6.5.2.3.3.1 KRB_AP_ERR_BADKEYVER after a Wake Up

This section addresses a scenario when an application server sends a Wake Up to a client and subsequently receives an AP Request that contains a ticket that is encrypted using an obsolete service key (results in the KRB_AP_ERR_BADKEYVER error code). This error normally requires the client to get another ticket and retry but in this particular scenario the client has to retry in the middle of a key management transaction.

In this scenario, the application server MUST reply to the invalid AP Request with the KRB_ERROR message with the KRB_AP_ERR_BADKEYVER error code. Subsequent to the reply, the server MUST wait for another AP Request and MUST use the same time out value that it would normally use when waiting for an AP Request. The client, upon getting back the above error code MUST attempt to obtain a new ticket from the KDC (if the client hasn't done so already while waiting for server's reply) and if successful, MUST send another AP Request to the application server. If the client is unsuccessful in obtaining another ticket, it MUST not reply. If the server times out waiting for the second AP Request, it MUST proceed as if it timed out waiting for the original AP Request.

If the application server is able to validate the second AP Request, it MUST then proceed as specified in Section 6.5.3. If the second AP Request again results in the KRB_AP_ERR_BADKEYVER error, the server MUST abort key management with this client and not reply.

6.5.2.3.3.2 KRB_AP_ERR_SKEW After a Wake Up

An application server is not required to check for a clock skew in this case, but if it does generate the KRB_AP_ERR_SKEW, the same procedure MUST be followed as in Section 6.5.2.3.3.1, except that the client MUST retry after adjusting the timestamp (see Section 6.5.2.3.2) instead of getting a new ticket.

6.5.3 Kerberized IPsec

This section specifies the Kerberized key management profile specific to IPsec ESP in transport mode. IPsec uses the term Security Association (SA) to refer to a set of security parameters. IPsec Security Associations are always uni-directional and they MUST always be established in pairs within PacketCable.

An MTA MUST establish SAs with the IP address from where the corresponding Kerberized IPsec key management message (AP-REP or REKEY) has been received. Note that a CMS can notify an MTA that it is listening for NCS messages on a different port. Also, both the CMS and the MTA can send NCS messages from different ports, and the response must be sent to the port from which the message was sent. Kerberized Key Management does not allow for the negotiation of source or destination ports. Therefore SAs established to protect NCS signaling need to support multiple ports. One way to accomplish this is to establish two separate policies, outbound and inbound, in the IPsec Security Policy Database (see [19]). The following table illustrates an example policy that would support changes in port numbers. Note that this table only illustrates inbound and outbound policies for NCS signaling between a specific MTA and a specific CMS. The table is not a complete IPsec Security Policy Database. Other entries would be required to support communications over different protocols with the same host (e.g., Kerberized Key Management), communications with other hosts, or default policies for unknown hosts.

Table 11. Example IPsec Security Policy Database Entries for NCS Signaling between MTA and CMS

Direction	Policy	Source IP	Source Port	Destination IP	Destination Port
Inbound – this applies to messages being received	Apply IPsec ESP	Remote IP address	Wildcard - any port	Local IP address	Bind to local port(s) that NCS messages will be sent from, and the provisioned NCS listening port.
Outbound – this applies to messages being sent	Apply IPsec ESP	Local IP address	Bind to local port(s) that messages will be sent from.	Remote IP address	Wildcard - any port

The DOI value for IPsec MUST be set to 1.

The ASD (Application-Specific Data) field in the AP Request key management message MUST be the SPI (Security Parameter Index) for the client's inbound Security Association. It is a 4-byte integer value, MSB first.

The ASD (Application-Specific Data) field in the AP Reply and Rekey key management messages MUST be the SPI (Security Parameter Index) for the server's inbound Security Association. It is a 4-byte integer value, MSB first

The subkey for IPsec MUST be a 46-byte value, defined as follows:

- If the AP-REQ does not include a subkey, the 46-octet subkey from AP-REP is taken as the subkey for IPsec.
- If the AP-REQ does include a subkey but no AP-REP (in the case of Rekey) is sent, then the 46-octet AP-REQ subkey is used as the subkey for IPsec.
- Otherwise, both the AP-REQ and the AP-REP messages include 46-octet subkeys, and their bit-by-bit XOR is the 46-byte subkey for IPsec.

An MTA MUST NOT perform Kerberized Key Management or establish IPsec Security Associations with a CMS when the pktcMtaDevCmsIpsecCtrl flag for that CMS is set to false in the pktcMtaDevCmsTable. Note that this flag may only be set in the MTA configuration file and cannot be updated using SNMPv3. In the case of an NCS Redirect or any other dynamic method for associating a new CMS with an MTA endpoint where there is not an entry in the pktcMtaDevCmsTable for the new CMS, the MTA MUST perform Kerberized Key Management and establish IPsec Security Associations with the new CMS.

The CMS MUST be capable of disabling its Kerberized Key Management interface. The CMS MUST NOT perform Kerberized Key Management or establish IPsec Security Associations when so configured.

6.5.3.1 Derivation of IPsec Keys

After the Application Server sends out an AP Reply message, it is ready to derive a new set of IPsec keys. Similarly, after the Client receives this AP Reply, it is ready to derive the same set of keys for IPsec. This section specifies how the IPsec keys are derived from the Kerberos subkey.

The size of the Kerberos subkey MUST be 46 bytes (the same as with the SSL or TLS pre-master secret).

The IPsec ESP keys MUST be derived in the following order:

1. Message authentication key for Client->Application Server messages
2. Encryption key for Client->Application Server messages
3. Message authentication key for Application Server->Client messages
4. Encryption key for Application Server->Client messages

For specific authentication and encryption algorithms that may be used by PacketCable for IPsec, refer to Section 6.1.2.

The derivation of the required keying material MUST be based on running a one-way pseudo-random function F(S, "IPsec Security Association") recursively until the right number of bits has been generated. Here, S is the Kerberos subkey and the ASCII string "IPsec Security Association" is taken without quotes and without a terminating null character. F is defined in Section 9.6.

6.5.3.2 Periodic Re-establishment of IPsec Security Associations

An IPsec SA is defined with an expiration time T_{EXP} and a grace period GP_{IPsec}. The subsections below specify how both the Client and the Application Server handle the re-establishment of IPsec Security Associations (re-establish flag was TRUE in the AP Reply). When the re-establishment of IPsec SAs is required there MUST always be at least one SA available for each direction and there MUST NOT be an interruption in the call signaling.

6.5.3.2.1 Periodic Re-establishment of IPsec SAs at the Client

If the re-establish flag is set, the Client MUST attempt to establish a new set of IPsec SAs (one for each direction) starting at the time T_{EXP} - GP_{IPsec}. At this time, the Client MUST send an AP Request as specified in Section 6.5. The destination IP Address of the AP Request message MUST be the destination IP Address of the outbound IPsec SA that is about to expire. After the Client receives an AP Reply, it MUST perform the following steps:

1. Create new IPsec SAs, based on the negotiated ciphersuite, SPIs and on the established Kerberos subkey, from which the IPsec keys are derived as specified in Section 6.5. The expiration time for the outgoing SA MUST be set to T_{EXP}, while the expiration time for the incoming SA MUST be set to T_{EXP} + GP_{IPsec}.
2. From this point forward, the new SA MUST be used for sending messages to the Application Server. The old SA that the Client used for sending signaling messages to the Application Server MAY be explicitly removed at this time, or it MAY be allowed to expire (using an IPsec timer) at the time T_{EXP}.

3. Continue accepting incoming signaling messages from the Application Server on both the old and the new incoming SAs, until the time $T_{EXP} + GP_{IPsec}$. After this time, the old incoming SA MUST expire. If a Client receives a signaling message from the Application Server using a new incoming SA at an earlier time, it MAY at that time remove the old incoming SA.

If the client fails to get any reply from the server and has to retry one or more times with another AP Request, the re-establish flag MUST be set to FALSE in each retry. This implies that when CMS processes a retry, it will remove any existing outgoing IPsec SAs, including the ones that may have been created after the processing of the initial AP Request, and proceed as if it is processing the SAs on demand (see Section 6.5.3.5.1).

6.5.3.2.2 Periodic Re-establishment of IPsec SAs at the Application Server

When an AP Request message is received with re-establish flag set, the Application Server MUST perform the applicable processing steps in Section 6.5.2. If the client is an MTA, the Application Server MUST also verify that the source IP address in the received datagram of the AP Request message is the same IP address as was used when the initial SA was established. The Application Server MUST ignore the AP Request if the IP addresses do not match.

In addition, the Application Server MUST perform the following steps, in the specified order, immediately before an AP Reply is returned.

1. Create new IPsec SAs, based on the negotiated ciphersuite, SPIs and on the established Kerberos subkey, from which the IPsec keys are derived as specified in Section 6.5.
2. Send back an AP Reply.
3. Continue sending signaling messages to the Client using an old outgoing SA until the time T_{EXP}. During the same period, accept incoming messages from either the old or the new incoming SA.
4. At the time T_{EXP} both the old incoming and the old outgoing SAs MUST expire. At the time T_{EXP}, the Application Server MUST switch to the new SA for outgoing signaling messages to the Client. If for some reason the new IPsec SAs were not established successfully, there would not be any IPsec SAs that are available after this time.

6.5.3.3 Expiration of IPsec SAs

An IPsec SA is defined with an expiration time T_{EXP} and a grace period $GP_{IPsec.}$ This section specifies how both the Client and the Application Server MUST handle the expiration of IPsec Security Associations (re-establish flag was FALSE in the AP Reply).

At the Client:

- Outgoing SA expires at T_{EXP}
- Incoming SA expires at $T_{EXP} + GP_{IPsec}$

At the Application Server:

- Outgoing SA expires at T_{EXP}
- Incoming SA expires at $T_{EXP} + GP_{IPsec}$

Whenever an IPsec SA has been expired and a signaling message needs to be sent by either the Client or the Application Server, the key management layer MUST be signaled to establish a new IPsec SA. It is established using the same procedures as the ones specified in Section 6.5.3.5.

6.5.3.4 Initial Establishment of IPsec SAs

When a Client is rebooted, it does not have any current IPsec SAs established with the Application Server, since IPsec SAs are not saved in non-volatile memory. In order to re-establish them, it MUST go through the recovery procedure that is described in Section 6.5.3.5.

6.5.3.5 On-demand Establishment of IPsec SAs

This section describes the recovery steps that MUST be taken in the case that a IPsec SA is somehow lost and needs to be re-established.

6.5.3.5.1 Client Loses an Outgoing IPsec SA

If a client attempts to send a signaling message to the Application Server without a valid IPsec SA, the IPsec layer in the Client will realize the SA is missing and return an error back to the signaling application. In this case, the following recovery steps MUST be taken at the key management layer:

1. The Client first makes sure that it has a valid Kerberos ticket for the Application Server. If not, it must first perform a PKINIT exchange as specified in Section 6.4.2.
2. Client sends a new AP Request to the Application Server and gets back an AP Reply, as specified in Section 6.5.2. After the receipt of an AP Reply the Client MUST:
    - create new IPsec SAs
    - remove any old outgoing IPsec SAs
    - be prepared to use both of the newly created IPsec SAs.
3. If the Kerberos ticket includes the optional caddr field and the caddr does not contain a matching source IP address for the AP Request datagram, the Application Server MUST ignore the request.
4. The Application Server MUST NOT set the ACK-required flag in the AP Reply. Right after sending out an AP Reply, the Application Server MUST be prepared to both send and receive messages on the newly created SAs.
5. After receiving this AP Request (with Re-establish flag = FALSE), the Application Server MUST remove any existing outgoing IPsec SAs that it might already have for this Client.

The key management application running on the Client MUST send an explicit signal to the signaling application when it completes the re-establishment of the IPsec SAs.

6.5.3.5.2 Client Loses an Incoming IPsec SA

When the Client receives an IP packet from an Application Server on an unrecognized IPsec SA, the Client MUST ignore this error and the packet MUST be dropped.

6.5.3.5.3 Application Server Loses an Outgoing IPsec SA

When an Application Server attempts to send a signaling message to the Client, and the IPsec layer in the Application Server realizes a valid SA is missing, the IPsec layer MUST return an error back to the signaling application.[3] In this case, the following recovery steps MUST be taken at the key management layer:

1. Application Server sends a Wake Up message to the Client.
2. The Client makes sure that it has a valid Kerberos ticket for the Application Server. If not, it MUST first obtain it from the KDC.
3. Client sends a new AP Request to the Application Server, as specified in Section 6.5.2. If the Kerberos ticket includes the optional caddr field and the caddr does not contain a matching source IP address for the AP Request datagram, the Application Server MUST ignore the request.
4. For each AP Request, the Client generates a nonce and puts it into the seq-number field. As specified in Section 6.5.2, the Client will save this nonce for a period of time specified by the pktcMtaDevCmsSolicitedKeyTimeout MIB object and wait for a matching AP Reply (this is not the same nonce as the Server-nonce received in the Wake Up). However, after this timeout, the

[3] In this case, there are no actual messages exchanged between the MTA and the CMS or other application server.

Client MUST NOT retry and MUST abort an attempt to establish a IPsec SA in response to a received Wake Up.

Once the Client gets back a matching AP Reply, it will be in the format specified in Section 6.5.2. The ACK-required flag in the AP Reply MUST be set, to insure that the Client replies with the SA Recovered message in the following step.

If this Client previously had any outgoing IPsec SAs with this Application Server IP address, they MUST be removed at this time. If the Client previously had a timer set for automatic refresh of IPsec SAs with this Application Server IP address, that automatic refresh MUST be reset or disabled. The Client MAY start using both of the newly created SAs. If the AP Reply had the Re-establish flag set, the Client MUST be prepared to automatically re-establish new IPsec SAs, as specified in Section 6.5.3.2.

The Application Server can receive signaling messages from the Client on the new incoming SA, but cannot yet start using an outgoing SA for sending messages to the Client.

5. Immediately after the Client establishes the new IPsec SAs, it MUST send a SA Recovered message to the Application Server.
6. Upon receipt of this message, the Application Server MUST immediately activate the new outgoing SA for sending signaling messages to the Client.

The key management application running on the Application Server MUST send an explicit signal to the signaling application when it completes the re-establishment of the IPsec SAs.

6.5.3.5.4 Application Server Loses an Incoming IPsec SA

When the Application Server receives an IP packet from a Client on an unrecognized IPsec SA, the Application Server MUST ignore this error and the packet MUST be dropped. In this case, any attempt at recovery (e.g., establishing a new SA) is prone to denial-of-service attacks.

6.5.3.6 IPsec-Specific Errors Returned in KRB_ERROR

Inside AppSpecificTypedData the oid field MUST be set to: enterprises (1.3.6.1.4.1) cableLabs (4491) clabProjects (2) clabProjPacketCable (2) pktcSecurity (4) errorCodes (1) ipSec (1).

The data-value field MUST correspond to the following typed-data value:

```
PktcKrbIpsecError ::= SEQUENCE {
  e-code [0] INTEGER,
  e-text [1] GeneralString OPTIONAL,
  e-data [2] OCTET STRING OPTIONAL
}
```

The e-code field MUST correspond to one of the following error code values:

KRB_IPSEC_ERR_NO_POLICY	1	No IPsec policy defined for request
KRB_IPSEC_ERR_NO_CIPHER	2	No support for requested ciphersuites
KRB_IPSEC_NO_SA_AVAIL	3	No IPsec SA available (i.e., SAD is full)
KRB_IPSEC_ERROR_GENERIC	16	Generic KRB IPsec error

The optional e-text field can be used for informational purposes (i.e., logging, network troubleshooting) and the optional e-data field is reserved for future use to transport any application data associated with a specific error.

6.5.4 Kerberized SNMPv3

This section specifies the Kerberized key management profile specific to SNMPv3, see [28]. In the case of SNMPv3, the security parameters are associated with the usmUserName (SNMPv3 user name), the agent's usmUserEngineID (SNMPv3 engine ID) and the manager's usmUserEngineID.

Multiple SNMP managers on different hosts but with the same user name are considered as unique Kerberos principals. Still, the SNMPv3 keys generated by any one of these SNMP managers MUST be shared across all the managers – as long as they apply to the same SNMPv3 user name and the same SNMPv3 engine ID (of the agent).

The security parameters consist of a single authentication key, a single privacy (encryption) key, SNMPv3 boot count and engine time. SNMPv3 privacy can be turned off by selecting a NULL encryption transform.

The DOI value for SNMPv3 MUST be set to 2.

The ASD field in the AP Request message MUST be set to the concatenation of the following:

Table 12. Required Format for Data in the AP Request

Attribute	Length
Agent's snmpEngineID Length	1 byte
Agent's snmpEngineID	variable
Agent's snmpEngineBoots	4 bytes, network byte order
Agent's snmpEngineTime	4 bytes, network byte order
usmUserName Length	1 byte
usmUserName	variable

The ASD field in the AP Reply message MUST be set to the concatenation of the following:

Table 13. Required Format for Data in the AP Reply

Attribute	Length
Manager's snmpEngineId Length	1 byte
Manager's snmpEngineId	variable
Manager's snmpEngineBoots	4 bytes, network byte order
Manager's snmpEngineTime	4 bytes, network byte order
usmUserName Length	1 byte
usmUserName	variable

For PacketCable MTAs, the usmUserName contains in it the MTA MAC address (see [4]). The manager MUST verify that this MAC address and the MTA FQDN specified in the MTA principal name match. The manager MUST also verify that any SNMP INFORM message containing a MAC address from the MTA contains a correct MAC address – the same one that is in the usmUserName. The usmUserName field inside the application-specific data field in the AP Reply MUST be the same as the one in the preceding AP Request.

The Rekey message is not used for SNMPv3 key management.

The subkey for SNMPv3 MUST be a 46-byte value.

6.5.4.1 Derivation of SNMPv3 Keys

After the server sends out an AP Reply message, it is ready to derive a new set of SNMPv3 keys. Similarly, after the client receives this AP Reply, it is ready to derive the same set of keys for SNMPv3. This section specifies how the SNMPv3 keys are derived from the Kerberos subkey.

The size of the Kerberos subkey MUST be 46 bytes.

The derived SNMPv3 keys MUST be as follows, in the specified order:

SNMPv3 authentication key

SNMPv3 privacy key

For specific authentication and encryption algorithms that may be used by PacketCable for SNMPv3, refer to Section 6.3.

The derivation of the required keying material MUST use a one-way pseudo-random function F(S, "SNMPv3 Keys") recursively until the right number of bits has been generated. Here, S is the subkey and the string "SNMPv3 Keys" is taken without quotes and without a terminating null character. F is defined in Section 9.6.

6.5.4.2 Periodic Re-establishment of SNMPv3 Keys

Periodic re-establishment of SNMPv3 keys, where the next set of keys is created before the old one expired, is currently not supported by PacketCable. The re-establish flag in the AP Reply key management message MUST be set to FALSE.

6.5.4.3 Expiration of SNMPv3 Keys

Expiration of SNMPv3 keys is currently not supported by PacketCable. The values of the Security Parameters Lifetime and Grace Period fields in the AP Reply MUST be set to 0.

6.5.4.4 Initial Establishment of SNMPv3 Keys

When a client is rebooted, it may not have any saved SNMPv3 keys established with the SNMP Manager. In order to re-establish them, it goes through the recovery procedure that is described in Section 6.5.4.5.1.

6.5.4.5 Error Recovery

This section describes the recovery steps that must be taken in the case that SNMPv3 keys are somehow lost and need to be re-established.

6.5.4.5.1 SNMP Agent Wishes to Send with Missing SNMPv3 Keys.

In the case of SNMP, an SNMP agent is not responsible for re-establishing SNMPv3 keys because it does not send unsolicited requests to the Provisioning Server after the initial provisioning is done. Still, an SNMP agent could attempt to re-establish SNMPv3 keys after it gets an SNMPv3 authentication error back from the SNMP manager. If the SNMP agent determines that it has incorrect SNMPv3 keys, it MUST perform the following steps before it is able to send out an SNMP message:

1. The agent first makes sure that it has a valid Kerberos ticket for the Application Server. If not, it must first obtain it as specified in Section 6.5.2.
2. The agent sends a new AP Request to the manager and gets back an AP Reply, as specified in Section 6.5.2. After the receipt of the AP Reply the agent is prepared to use the newly created SNMPv3 keys. In this scenario, the SNMP manager MUST NOT set an ACK-required flag in the AP Reply. Right after sending out an AP Reply, the manager is prepared to both send and receive messages with the new SNMPv3 keys. After receiving this AP Request (with Re-establish flag = FALSE), the manager MUST remove its previous set of SNMPv3 keys that it might already have for this agent (and for this SNMPv3 user name).

It is possible that the SNMP manager already initiated key management (with a Wake Up) but instead receives an unsolicited AP Request from the agent (with server-nonce = 0). This unlikely scenario might occur if the manager and the agent decide to initiate key management at about the same time. In this case, the SNMP manager MUST ignore the unsolicited AP Request message and continue waiting for the one that is in response to a Wake Up.

6.5.4.5.2 SNMP Agent Receives with Missing SNMPv3 Keys

If SNMP agent receives a request from SNMP manager and is unable to find SNMPv3 keys for the specified USM User Name, the agent MUST process the SNMP message according to [28], [38].

6.5.4.5.3 SNMP Manager Wishes to Send with Missing SNMPv3 Keys

SNMP manager attempts to send a message to the agent and does not find the desired user's SNMPv3 keys (or considers the existing SNMPv3 keys invalid or compromised). In this case, the following recovery steps MUST be taken at the key management layer:

1. Manager sends a Wake Up message to the agent.
2. The agent makes sure that it has a valid Kerberos ticket for the manager. If not, it MUST first obtain it from the KDC.
3. Agent sends a new AP Request to the manager, as specified in Section 6.5.2. For each AP Request, the agent generates a nonce and puts it into the seq-number field. As specified in Section 6.5.3.5.3, the agent will save this nonce for a period of time specified by the pktcMtaDevProvSolicitedKeyTimeout MIB object and wait for a matching AP Reply (this is not the same nonce as the server-nonce received in the Wake Up). However, after this timeout, the agent MUST NOT retry and MUST abort an attempt to establish SNMPv3 keys in response to a received Wake Up.

   Once the agent gets back a matching AP Reply, it will be in the format specified in Section 6.5.2. The ACK-required flag in the AP Reply MUST be set, to insure that the agent replies with the SA Recovered message in the following step.

   If this agent previously had SNMPv3 keys for the specified SNMPv3 user, they MUST be removed at this time.
4. After the receipt and validation of the AP Reply, the agent sends SA Recovered message to the manager. At this time the agent will be ready to use the new SNMPv3 keys and will enable SNMPv3 security.
5. Upon receipt of the SA Recovered message, the manager will immediately activate the new set of SNMPv3 keys and will enable SNMPv3 security.

It is possible that the SNMP agent already initiated key management (with an unsolicited AP Request) but instead receives a Wake Up from the manager. This unlikely scenario might occur if the manager and the agent decide to initiate key management at about the same time. In this case, the SNMP agent MUST abort waiting for the reply to the unsolicited AP Request message and instead generate a new AP Request in response to the Wake Up.

If an SNMP agent receives a second Wake Up message from a different SNMP manager (FQDN or IP address) before the first key management session has been completed, the SNMP agent MUST ignore the second Wake Up message.

6.5.4.6 SNMPv3-Specific Errors Returned in KRB_ERROR

Inside AppSpecificTypedData the oid field MUST be set to:

> enterprises (1.3.6.1.4.1) cableLabs (4491) clabProjects (2) clabProjPacketCable (2) pktcSecurity (4) errorCodes (1) snmpv3 (2).

The data-value field MUST correspond to the following typed-data value:

```
PktcKrbSnmpv3Error ::= SEQUENCE {
  e-code [0] INTEGER,
  e-text [1] GeneralString OPTIONAL,
  e-data [2] OCTET STRING OPTIONAL
}
```

The e-code field MUST correspond to one of the following error code values:

KRB_SNMPV3_ERR_USER_NAME	1	Unrecognized SNMPv3 user name
KRB_SNMPV3_ERR_NO_CIPHER	2	No support for requested ciphersuites

KRB_ SNMPV3_ERR_ENGINE_ID	3	Invalid SNMPv3 Engine ID Specified
KRB_ SNMPV3_ERROR_GENERIC	16	Generic KRB SNMPv3 error

The optional e-text field can be used for informational purposes (i.e., logging, network troubleshooting) and the optional e-data field is reserved for future use to transport any application data associated with a specific error.

6.6 End-to-End Security for RTP

RTP security is currently fully specified in Section 7.6.2.1. Key Management for RTP requires that both the (encryption) Transform ID and the Authentication Algorithm are specified, analogous to the IPsec key management. This section lists the Transform IDs and Authentication Algorithms that are available for RTP security.

Table 14. RTP Packet Transform Identifiers

Transform ID	Value	Key Size (in bits)	MUST Support	Description
RTP_ENCR_NULL	0x50	N/A	yes	Encryption turned off
RTP_AES	0x51	128	yes	AES-128 in CBC mode with 128-bit block size
RTP_XDESX_CBC	0x53	192	no	DESX-XEX-CBC
RTP_DES_CBC_PAD	0X54	128	no	DES-CBC-PAD
RTP_3DES_CBC	0X56	128	no	3DES-EDE-CBC
reserved	0x57-59	-	-	

The RTP_AES and RTP_ENCR_NULL Transform IDs MUST be supported. AES-128 [35] MUST be used in CBC mode with a 128-bit block size and an Initialization Vector (IV) generated in accordance with Section 7.6.2.1.2.2.2. AES-128 requires 10 rounds of cryptographic operations [35].

Table 15. RTP PacketCable Authentication Algorithms

Authentication Algorithm	Value	Key Size (in bits)	MUST Support	Description
AUTH_NULL	0x60	0	yes	Authentication turned off.
reserved	0x61	-	-	
RTP_MMH_2	0x62	variable (see Section 7.6.2.1.2.1.1)	yes	2-byte MMH MAC
reserved	0x63	-	-	
RTP_MMH_4	0x64	variable (see Section 7.6.2.1.2.1.1)	yes	4-byte MMH MAC
reserved	0x65	-	-	

The Authentication Algorithms AUTH_NULL, RTP_MMH_2 and RTP_MMH_4 MUST be supported.

6.7 End-to-End Security for RTCP

RTCP security is currently fully specified in Section 7.6.2.2. Key Management for RTCP requires that both the (encryption) Transform ID and the Authentication Algorithm be specified. This section lists the Transform IDs and Authentication Algorithms that are available for RTCP security.

Table 16. RTCP Packet Transform Identifiers

Transform ID	Value	Key Size (in bits)	MUST Support	Description
RTCP_ENCR_NULL	0x70	0	yes	Encryption turned off.
AES-CBC	0x71	128	yes	AES-128 in CBC mode with 128-bit block size
XDESX-CBC	0x72	192	no	DESX-XEX-CBC
DES-CBC-PAD	0x73	128	no	DES-CBC-PAD
3DES-CBC	0x74	128	no	3DES-EDE-CBC
reserved	0x75-7f	-	-	

The AES-CBC and RTCP_ENCR_NULL Transform IDs MUST be supported. AES-128 [35] MUST be used in CBC mode with a 128-bit block size and a randomly generated Initialization Vector (IV). AES-128 requires 10 rounds of cryptographic operations [35].

Table 17. RTCP Authentication Algorithms

Transform ID	Value	Key Size (in bits)	MUST Support	Description
RTCP_AUTH_NULL	0x80	N/A	yes	Authentication turned off
HMAC-SHA1-96	0x81	160	yes	First 12 bytes of the HMAC-SHA1 as described in [23].
HMAC-MD5-96	0x82	128	no	First 12 bytes of the HMAC-MD5 as described in [37].
reserved	0x83-8f	-	-	

The HMAC-SHA1-96 and RTCP_AUTH_NULL authentication algorithm MUST be supported

6.8 BPI+

All E-MTAs and S-MTAs MUST use DOCSIS 1.1 compliant cable modems that implement BPI+ [9]. Baseline Privacy Plus (BPI+) provides security services to the DOCSIS 1.1 data link layer traffic flows running across the cable access network, i.e., between CM and CMTS. These services are message confidentiality and access control. The BPI+ security services operating in conjunction with DOCSIS 1.1 provide cable modem users with data privacy across the cable network and protect cable operators from theft of service.

The protected DOCSIS 1.1 MAC data communications services fall into three categories:

- Best-effort, high-speed, IP data services;
- QoS (e.g., constant bit rate) data services; and
- IP multicast group services.

When employing BPI+, the CMTS protects against unauthorized access to these data transport services by (1) enforcing encryption of the associated traffic flows across the cable network and (2) authenticating the DOCSIS MAC management messages that CMs use to establish QoS service flows. BPI+ employs a client/server key management protocol in which the CMTS (the server) controls distribution of keying material to client CMs. The key management protocol ensures that only authorized CMs receive the encryption and authentication keys needed to access the protected services.

Baseline Privacy Plus has two component protocols:

- An encapsulation protocol for encrypting packet data across the cable network. This protocol defines (1) the frame format for carrying encrypted packet data within DOCSIS MAC frames, (2) a set of

supported *cryptographic suites*, i.e., pairings of data encryption and authentication algorithms, and (3) the rules for applying those algorithms to a DOCSIS MAC frame's packet data.

- A key management protocol (Baseline Privacy Key Management, or "BPKM") provides the secure distribution of keying data from CMTS to CMs. Through this key management protocol, CM and CMTS synchronize keying data; in addition, the CMTS uses the protocol to enforce conditional access to network services.

Baseline Privacy Plus does not provide any security services beyond the DOCSIS 1.1 cable access network. The majority of PacketCable's signaling and media traffic flows, however, take paths that traverse the managed IP "back haul" networks, which lie behind CMTSs. Since DOCSIS and PacketCable service providers typically will not guarantee the security of their managed IP back haul networks, the PacketCable security architecture defines end-to-end security mechanisms for all these flows. End-to-end security is provided at the Network layer through IPsec, or, in the case of Client media flows, at the application/transport layer through RTP application layer security. Thus, PacketCable does not rely on BPI+ to provide security services to its component protocol interfaces.

6.9 TLS

6.9.1 Overview

The TLS protocol [17] provides privacy and data integrity over a reliable transport layer protocol such as TCP. The protocol is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. The TLS Record Protocol is used to securely encapsulate upper layer protocols, while the TLS Handshake Protocol provides the key management functionality required to establish TLS sessions.

In PacketCable, TLS is used to secure SIP based signaling between SIP endpoints such as the CMS and EBPs.

6.9.2 PacketCable Profile for TLS with SIP

Unless specified within this document, PacketCable SIP interfaces requiring TLS MUST be compliant with the TLS specification [17] and any requirements specified in [40] relating to its usage in SIP.

TLS [17] supports the negotiation and use of compression methods. However, since these methods are not specified within TLS [17], compression MUST NOT be used in PacketCable.

6.9.2.1 TLS Ciphersuites

In TLS, the ciphersuite includes the authenticated key agreement (AKE) method used in the TLS handshake, as well as encryption and authentication ciphers used to secure the record layer. Ciphersuites are negotiated with the TLS client presenting a list of supported ciphersuites in the Client Hello message, and the server responding with the selected ciphersuite in the Server Hello message.

The following table describes the TLS ciphersuites defined in [17] and [41] supported by PacketCable:

Table 18. TLS Ciphersuites

TLS Ciphersuite	Support	AKE Method	Encryption	Auth.
TLS_RSA_WITH_AES_128_CBC_SHA	MUST	RSA	AES-128 CBC	SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	MUST	Ephemeral Diffie-Hellman with RSA signatures	AES-128 CBC	SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SHOULD	RSA	3DES CBC	SHA

TLS Ciphersuite	Support	AKE Method	Encryption	Auth.
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	SHOULD	Ephemeral Diffie-Hellman with RSA signatures	3DES CBC	SHA

6.9.2.2 PacketCable TLS Certificates

TLS is a client-server based protocol with optional client authentication. However, in PacketCable, mutual authentication using RSA based certificates MUST be used. The TLS server MUST send a Certificate Request to the client. Both the TLS client and server certificates MUST conform to the PacketCable Server Certificate as specified in Section 8.2.3.4.4.

PacketCable Server Certificates include a server identifier (based on FQDN or IP address) embedded within the CN of the Subject Name field. Before accepting or continuing with a TLS connection, the TLS server or client MUST validate the remote server identifier to ensure it matches the IP address used for the TCP/TLS connection, in addition to any other local policy (i.e. provisioned list of allowed remote TLS endpoints based on FQDN or IP address).

In addition to the CableLabs Service Provider Root certificate, a TLS implementation MAY support a list of trusted CAs (Certificate Authorities) to facilitate inter-working between PacketCable domains (i.e. between Server Providers).

6.9.2.3 Connection Persistence and Re-Use

Since TCP connection and TLS session establishment (which relies on TCP) can be quite costly both in terms of performance and network latency, they are not suited for on-demand SIP signaling. As such, TLS sessions SHOULD be kept persistent as much as possible and SIP connection re-use SHOULD be supported.

6.9.2.4 Session Caching

In TLS, it is possible to resume a previous session if it has been cached on both the TLS client and server. Resuming sessions drastically speeds up the session establishment, as fewer messages are exchanged and authentication is based on symmetric key cryptography.

In PacketCable, TLS session caching SHOULD be supported. A TLS client initiating a TLS session MUST attempt to resume a cached session if it has retained a session for the remote server. The duration for which a TLS client or server must retain a cached session is a local policy and implementation specific.

7 SECURITY PROFILE

The PacketCable architecture defines over half a dozen networked components and the protocol interfaces between them. These networked components include the media terminal adapter (MTA), call management server (CMS), signaling gateway (SG), media gateway (MG) and a variety of OSS systems (DHCP, TFTP and DNS servers, network management systems, provisioning servers, etc.). PacketCable security addresses the security requirements of each constituent protocol interface by:

- Identifying the threat model specific to each constituent protocol interface
- Identifying the security services (authentication, authorization, confidentiality, integrity, non-repudiation) required to address the identified threats
- For each constituent protocol interface, specifying the particular security mechanism providing the required security services

Section 5.2 summarizes the threat models applicable to PacketCable's protocol interfaces. In this section, we identify the security service requirements of each protocol interface and security mechanisms providing those services.

The security mechanisms include both the security protocol (e.g., IPsec, RTP-layer security, SNMPv3 security) and the supporting key management protocol (e.g., IKE, PKINIT/Kerberos).

The security analysis in Section 5.3.3 is organized by functional categories. For each functional category, we identify the constituent protocol interfaces, the security services required by each interface, and the particular security mechanism employed to deliver those security services. Each per-protocol security description includes the detailed information sufficient to ensure interoperability. This includes cryptographic algorithms and cryptographic parameters (e.g., key lengths).

As a convenient reference, each functional category's security analysis includes a summary security profile matrix of the following form (Media security profile matrix shown):

Table 19. RTP – RTCP Security Profile Matrix

	RTP (MTA – MTA, MTA – PSTN GW)	**RTCP (MTA – MTA, MTA – MG, MG – MG)**
authentication	optional (indirect)	optional (indirect)
access control	optional	optional
integrity	optional	yes
confidentiality	yes	yes
non-repudiation	no	no
Security mechanisms	Application Layer Security via RTP PacketCable Security Profile keys distributed over secured MTA-CMS links AES-128 encryption algorithm Optional 2-byte or 4-byte MAC based on MMH algorithm PacketCable supports ciphersuite negotiation.	Application Layer Security via RTCP PacketCable Security Profile keys distributed over secured MTA-CMS links RTCP ciphersuites are negotiated separately from the RTP ciphersuites and include both encryption and message authentication algorithms. Keys are derived from the end-end secret using the same mechanism as used for RTP encryption.

Each matrix column corresponds to a particular protocol interface. All but the last row corresponds to a particular security service; the cell contents in these rows indicate whether the protocol interface requires the corresponding security service. The final row summarizes the security mechanisms selected to provide the required services.

Note that the protocol interface column headings not only identify the protocol, but also indicate the network components the protocols run between.

7.1 Device and Service Provisioning

Device provisioning is the process by which an MTA is configured to support voice communications service. The MTA provisioning process is specified in [4].

The following figure illustrates only the flows involved with the Secure provisioning processes. The provisioning specification lays out in detail these Secure Provisioning flows along with two non-secure MTA provisioning flows called Basic and Hybrid. The Secure Provisioning flows involving security mechanisms are described in this section of the document. Refer to the provisioning specification for the non-secure flows [4].

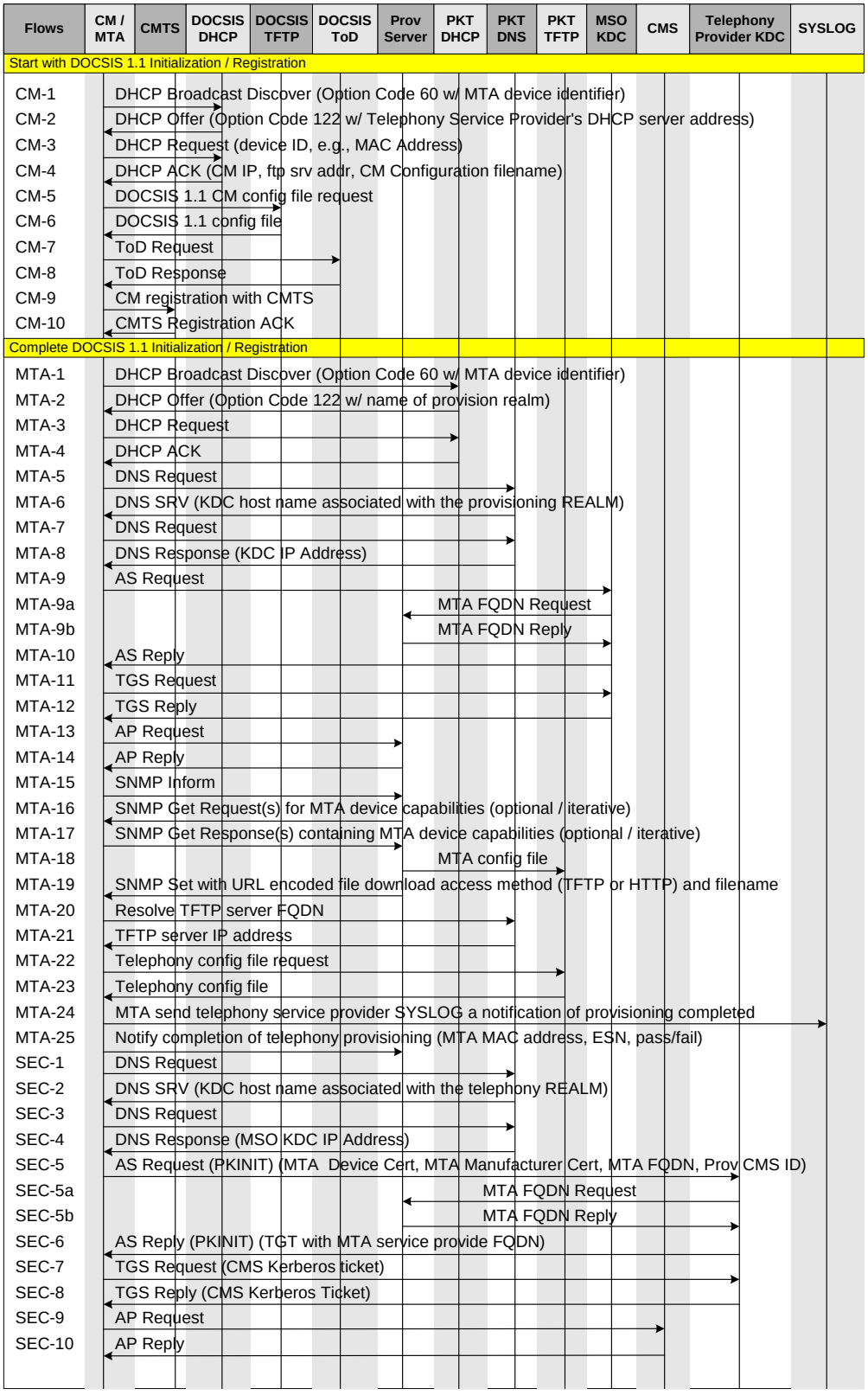


Figure 10. PacketCable Provisioning Flows

As part of the provisioning process, the MTA performs Kerberos key management (AS Request/AS Reply and AP Request/AP Reply, and optional TGS Request/TGS Reply).

The following table describes the execution of the Kerberos key management step during MTA Provisioning:

Table 20. Kerberos Key Management During MTA Provisioning

Flow Step	Security Requirement	Life Time	Step Bypass Permitted
MTA-9/MTA-10 – AS Request/AS Reply (see Section 6.4.1)	TGT ticket if using TGS Request, Provisioning Server Ticket if otherwise	Max. 7 days	This step MUST NOT be performed if the MTA already possesses a valid ticket for the Provisioning Server.
MTA-11/MTA 12 – TGS Request/TGS Reply (see Section 6.4.4)	Applies when a TGT is used. Obtains a Provisioning Server Ticket.	Lifetime set to expire no later than the expiration time of the TGT ticket	This step MUST NOT be performed if the MTA already possesses a valid ticket for the Provisioning Server.
MTA-9a/MTA-9b – MTA FQDN Request/MTA FQDN Reply (see Section 6.4.7)	MTA FQDN Request and Reply are protected using Kerberos tickets		These steps will not occur if MTA-9 is skipped. Otherwise, this step cannot be bypassed.
MTA-13/MTA-14 – AP Request/AP Reply (see Section 6.5.2 and Section 6.5.4)	Initial SNMPv3 authentication and privacy keys for the MTA. The user name for the MTA is specified as "MTA-Prov-xx:xx:xx:xx:xx:xx". Where xx:xx:xx:xx:xx:xx represents the MAC address of the MTA. AP Req /AP Rep messages don't specify the SNMPv3 key expiration time in the protocol, but the SNMP Manager may still set up expiration time locally; after the keys expire the manager can send a Wake Up message to create a new set of SNMPv3 keys.	Expiration is not supported by PacketCable.	None - new SNMPv3 keys and User Ids are created each time the MTA is reinitialized. It is assumed that SNMPv3 keys and User Ids are not saved in NVRAM. Also note that this step is used for Engine ID determination and SNMPv3 time synchronization - the two sides exchange initial values for SNMPv3 boots and engine time parameters.

An MTA MUST get a new ticket before performing Kerberized Key Management with a particular Application Server if the ticket(s) it currently possesses is not valid. A ticket would no longer be valid if the KDC REALM or Application Server FQDN changes, if the MTA's IP address has changed, or if the current time, adjusted by the time offset for that REALM or Application Server, does not fall within the ticket validity period.

The $PKINIT_{GP}$ for the Provisioning Server's realm is specified in the MTA MIB inside the realm table. When the MTA implementation requests a TGT in an AS Request and when the MTA needs to obtain tickets for one or more CMSs in the same realm as the Provisioning Server, the $PKINIT_{GP}$ value specified in the MIB MUST be used to refresh the TGT. In all other cases, the AS Request for the TGT in the Provisioning Server's realm or for the Provisioning Server's ticket directly MAY be issued on-demand.

The TGS Grace Period is not specified for the key management between the MTA and the Provisioning Server. The TGS Request for the Provisioning Server's ticket MAY be issued on-demand.

7.1.1 Device Provisioning

Device provisioning occurs when an MTA device is inserted into the network. A provisioned MTA device that is not yet associated with a billing record MAY have minimal voice communications service available.

Device provisioning involves the MTA making itself visible to the network, obtaining its IP configuration and downloading its configuration data.

As defined in [4], the PacketCable architecture supports three provisioning flow:

- Basic Flow
- Hybrid Flow
- Secure Flow

The Basic and Hybrid Flows are completely insecure flows (i.e. there are no mechanisms in the flows that would prevent a user from provisioning their own MTA). The Basic and Hybrid Flows also do not provide a means to secure the SNMP management interface on the MTA. Service providers that choose to deploy MTAs with one of these insecure flows must accept that there are security risks. For example, a Denial-of-Service attack could be mounted by sending SNMP TRAPs and INFORMs to the MSO's management system. The management system would have to process them, even though they are unauthenticated. Unfortunately, the inclusion of these insecure flows also poses security risks for Service Providers that choose to deploy MTAs with the Secure Flow.

MTAs that support the insecure flows may be provisioned by a user, even if the service provider is using the Secure Flows. Unauthorized provisioning of an MTA allows a user to provide their own configuration file. The MTA could then be used to communicate normally with a CMS. Alternatively, un-authorized provisioning of an MTA could be used to bypass service provider controls on secure software download (in the case of the S-MTA) and provide a software image that has some perceived value (such as a security vulnerability).

With respect to the Secure Flow, support for SNMPv2c coexistence for network management operations also introduces vulnerabilities to service providers that use the Secure Flow (unauthenticated TRAPs and INFORMs could be sent). The best way to address these vulnerabilities is to disable SNMPv2c coexistence.

Therefore it is recommended, as always, that service providers use multiple layers of security to ensure that their CMSs and back-office systems are protected against rogue MTAs.

7.1.1.1 Security Services

7.1.1.1.1 MTA-DHCP Server

Authentication and Message Integrity is desirable on this interface, in order to prevent denial-of-service attacks, that cause an MTA to be improperly configured. Securing DHCP is considered an operational issue to be evaluated by each network operator. It is possible to use access control through the local DHCP relay inside the local loop. IPsec can be used for security between the DHCP relay and the DHCP server.

7.1.1.1.2 MTA-SNMP Manager

This section applies to all SNMPv3 messages between the MTA and an SNMPv3 Manager. Within the PacketCable architecture, the Provisioning Server includes the SNMPv3 Manager function, although SNMPv3 traffic occurs both during and after the provisioning phase.

Authentication: the identity of the MTA that is sending configuration parameters and faults to the SNMP manager must be authenticated, to prevent denial of service attacks. For example, the Provisioning Server may be tricked into continuously creating bogus configuration files or into creating a configuration file based on incorrect MTA capabilities that in effect disable that MTA.

Also, during the provisioning sequence the MTA is told (via an SNMP Set) the parameters needed to find, authenticate and decrypt its configuration file. If this SNMP Set were forged, it would disrupt the MTA provisioning sequence.

Message Integrity: required to prevent denial of service attacks at the OSS and at the MTA – see the above description of the denial of service attacks under authentication.

Confidentiality: may be used to protect sensitive MTA configuration data. PacketCable currently does not specify any such sensitive MTA parameters and so confidentiality is optional.

Access Control: write access to the MTA configuration parameters must be allowed only to the authorized OSS users, to prevent denial of service/misconfiguration attacks. Read access can be enforced in conjunction with confidentiality, which is optional (see above on confidentiality).

Note that DHCP is used to configure the MTA with the Kerberos realm name, which points it to a particular KDC. DHCP also configures the MTA with the location of the Provisioning Server. Since PacketCable currently does not specify DHCP security, by faking DHCP responses it is possible to point MTAs to a wrong Provisioning Server and to a wrong KDC that permits security establishment with that Provisioning Server. (The MTA would only authenticate that wrong KDC if the CableLabs Service Provider Root CA signed the KDC certificate.) So, it is possible to bypass access control, but the attack has to be orchestrated by another MSO that had also been certified by PacketCable.

7.1.1.1.3 MTA-Provisioning Server, via TFTP Server

Authentication: required to prevent denial-of-service attacks that cause an MTA to be improperly configured.

Message Integrity: required to prevent denial-of-service attacks that cause an MTA to be either improperly configured or configured with old configuration data that was replayed.

Confidentiality: optional, it is up to the Provisioning Server to decide whether or not to encrypt the file.

Access Control: not required at the TFTP Server. If needed, MTA configuration file is encrypted with the Provisioning Server-MTA shared key.

Non-Repudiation: not required.

7.1.1.2 Cryptographic Mechanisms

7.1.1.2.1 Call Flows MTA-15, 16, 17: MTA-SNMP Manager: SNMP Inform/Get Requests/Responses

All SNMP traffic between the MTA and the SNMP Manager in both directions is protected with SNMPv3 security [28] during the Secure Provisioning process. PacketCable requires that SNMPv3 message authentication is always turned on with privacy being optional (see Section 6.3). The only SNMPv3 encryption algorithm is currently DES-CBC. This is the limitation of the SNMPv3 IETF standard, although stronger encryption algorithms are desirable. See Section 6.3 for the list of SNMPv3 cryptographic algorithms supported by PacketCable.

7.1.1.2.2 Call Flow MTA-18: Provisioning Server-TFTP Server: Create MTA Config File

This section describes the MTA Config file creation in the Secure Provisioning Flow. In this flow, the Provisioning Server builds an MTA device configuration file. This file MUST contain the following configuration info for each endpoint (port) in the MTA:

- CMS name (FQDN format)
- Kerberos Realm for this CMS
- Telephony Service Provider Organization Name
- PKINIT Grace Period

This file MUST be authenticated and MAY be encrypted. If the configuration file is encrypted then the SNMPv3 privacy MUST be used in order to transport the configuration file encryption key securely. Once the Provisioning Server builds the configuration file, it will perform the following steps:

1. The Provisioning Server decides to encrypt the file, it creates a configuration file encryption key and encrypts the file with this key. The encryption algorithm MUST be the same as the one that is used for SNMPv3 privacy. It then stores the key and the cipher. The file MUST be encrypted using the following procedure:
    a. prepend the file contents with a random byte sequence, called a confounder. The size of the confounder MUST be the same as the block size for the encryption algorithm. In the case of DES it is 8 bytes.
    b. append random padding to the result in (a). The output of this step is of length that is a multiple of the block size for the encryption algorithm.
    c. encrypt the result in (b) using IV=0. The output of this step is the encrypted configuration file.
2. It creates a SHA-1 hash of the configuration file and stores it. If the file was encrypted, the hash is taken over the encrypted file.
3. It sends the following items to the MTA in the SNMP SET in the flow MTA-19.
    a. pktcMtaDevConfigKey, which is the configuration file encryption key MIB variable generated in step 1.
    b. pktcMtaDevConfigHash, which is the SHA-1 of the configuration file MIB variable generated in step 2.
    c. Name and location of the configuration file.

Steps 1 and 2 MUST occur only when a configuration file is created or an existing file is modified. If the pktcMtaDevConfigKey is set, then the MTA MUST use this key to decrypt the configuration file. Otherwise, MTA MUST assume that the file is not encrypted. SNMPv3 provides authentication when the pktcMtaDevConfigHash is set and therefore the configuration file is authenticated indirectly via SNMPv3.

In the event that SNMPv3 privacy is selected during the key management phase, but is using a different algorithm than the one that was selected to encrypt the configuration file (or the configuration file was previously in the clear), the configuration file MUST be re-encrypted and the TFTP server directory MUST be updated with the new file. Similarly, if the Provisioning Server decides not to encrypt the file this time, after it was previously encrypted, the TFTP server directory MUST be updated with the new file.

MTA endpoints MAY also be configured for IP Telephony service while the MTA is operational. In that case the same information that is normally assigned to an endpoint in a configuration file MUST be assigned with SNMP Set commands.

7.1.1.2.3 Call Flows MTA-19, 20 and 21: Establish TFTP Server Location

This set of call flows is used to establish the IP address of the TFTP server from where the MTA will retrieve its configuration file. Although flow MTA-19 is authenticated via SNMPv3, MTA-20 and 21 are not authenticated.

Flow MTA-21 allows for denial-of-service attacks, where the MTA is pointed to a wrong TFTP server (IP address). The MTA cannot be fooled in accepting the wrong configuration file since checking the hash of the file authenticates the file – this denial-of-service attack will result in failed MTA provisioning.

The denial-of-service threats, where responses to DNS queries are forged, are currently not addressed by PacketCable. It is mainly because DNS security (DNSSEC) is not yet available as a commercial product and would cause significant operational difficulty in the conversion of the DNS databases.

7.1.1.2.4 Call Flows MTA-22, 23: MTA-TFTP Server: TFTP Get/Get Response

The TFTP get request is not authenticated and thus anyone can request an MTA configuration file. This file does not contain any sensitive data and may be encrypted with the Provisioning Server-MTA shared key if the Provisioning Server chooses to. In this case no one except the MTA can make use of this file.

This flow is open for a denial-of-service attack, where the TFTP server is made busy with useless TFTP-get requests. This denial-of-service attack is not addressed at this time.

The TFTP get response retrieves a configuration file from the TFTP server. The contents of the configuration file are listed in 7.1.1.2.2.

7.1.1.2.5 Security Flows

For each CMS specified in the pktcMtaDevCmsTable table with pktcMtaDevCmsIpsecCtrl value set to true and assigned to a provisioned MTA endpoint, the MTA MUST perform the following security flows after the provisioning process and prior to any NCS message exchange. For each CMS specified in pktcMtaDevCmsTable with pktcMtaDevCmsIpsecCtrl set to false, the MTA MUST NOT perform the following flows and MUST send and receive NCS messages without IPsec (i.e., NCS packets are sent in the "clear").

Table 21. Post-MTA Provisioning Security Flows

Sec Flow	Flow Description	If Step Fails, Proceed Here
Get Kerberos tickets associated with each CMS with which the MTA communicates.		
SEC-1	DNS SRV Request The MTA requests the Telephony KDC host name for the Kerberos realm. This step MUST NOT be performed if the MTA already possesses a valid ticket for the CMS.	SEC-1
SEC-2	DNS SRV Reply Returns the Telephony KDC host name associated with the provisioning REALM. If the KDC's IP Address is included in the Reply, proceed to SEC-5. This step MUST NOT be performed if the MTA already possesses a valid ticket for the CMS.	SEC-1
SEC-3	DNS Request The MTA now requests the IP Address of the Telephony KDC. This step MUST NOT be performed if the MTA already possesses a valid ticket for the CMS.	SEC-1
SEC-4	DNS Reply The DNS Server returns the IP Address of the Telephony KDC. This step MUST NOT be performed if the MTA already possesses a valid ticket for the CMS.	SEC-1
SEC-5	AS Request For each different CMS assigned to voice communications endpoints, the MTA requests a TGT or a Kerberos Ticket for the CMS by sending a PKINIT REQUEST message to the KDC. This request contains the MTA Device Certificate and the MTA FQDN. This step MUST NOT be performed if the MTA already possesses a valid ticket for the CMS.	Report alarm. Abort establishment of signaling security.
SEC-5a	MTA FQDN Request The KDC requests the MTA's FQDN from the Provisioning Server. This step will not occur if the MTA skips SEC-5.	

Sec Flow	Flow Description	If Step Fails, Proceed Here
SEC-5b	MTA FQDN Reply The Provisioning Server replies to the KDC request with the MTA's FQDN. This step will not occur if the MTA skips SEC-5.	
SEC-6	AS Reply The KDC sends the MTA a PKINIT REPLY message containing the requested Kerberos Ticket. This step MUST NOT be performed if the MTA already possesses a valid ticket for the CMS.	Proceed to SEC-5 or abort signaling security depending upon error conditions.
SEC-7	TGS Request In the case where the MTA obtained a TGT in SEC-6, it now obtains the Kerberos ticket for the TGS request message. This step MUST NOT be performed if the MTA already possesses a valid ticket for the CMS	Report alarm. Abort establishment of signaling security.
SEC-8	TGS Reply Response to TGS Request containing the requested CMS Kerberos Ticket. This step MUST NOT be performed if the MTA already possesses a valid ticket for the CMS	Proceed to SEC-7/SEC-5 or abort signaling security depending upon error conditions.
SEC-9	AP Request The MTA requests a pair of IPsec simplex Security Associations (inbound and outbound) with the assigned CMS by sending the assigned CMS an AP REQUEST message containing the CMS Kerberos Ticket.	Report alarm. Abort establishment of signaling security.
SEC-10	AP Reply The CMS establishes the Security Associations and then sends an AP REPLY message with the corresponding IPsec parameters. The MTA derives IPsec keys from the subkey in the AP Reply and establishes IPsec SAs.	Proceed to SEC-9/SEC-7/SEC-5 or abort signaling security depending upon error conditions.

Several tables in the MTA MIB are used to control security flows SEC-1 through SEC-10 (see Table 21).

The CMS table (pktcMtaDevCmsTable) and the realm table (pktcMtaDevRealmTable) are used for managing the MTA security signaling. The realm table defines the domains for the CMSs. The CMS table defines the CMSs within the domains. An endpoint is associated with one CMS at any given time. The following restrictions MUST be adhered to:

a. The realm table in the configuration file MUST at a minimum include an entry for the realm that is identified in DHCP option 122, suboption 6.

b. There MUST be a realm table entry for each CMS table entry. Multiple CMS table entries MAY utilize the same realm table entry.

c. Each MTA endpoint defined in the NCS endpoint table (pktcNcsEndPntConfigTable) MUST be configured with a CMS FQDN (pktcNcsEndPntConfigCallAgentId) that is also present in the CMS table (pktcMtaDevCmsFqdn).

d. All members of a CMS cluster defined by the same FQDN MUST use the same configuration for establishing Security Associations as defined in pktcMtaDevCmsTable.

e. If NCS signaling selects a CMS (with an N: parameter selection) that is not defined by an entry in the CMS table, the same realm and CMS parameters, with the exception of the CMS FQDN and

pktcMtaDevCmsIpsecCtrl, are used as defined in the current CMS table entry. The pktcMtaDevCmsIpsecCtrl flag for the new CMS MUST be set to true.

The use of the security-relevant MIB tables immediately following step MTA-25 is as follows:

1. The MTA finds a list of CMSs with which it needs to establish IPsec SAs. This list MUST include every CMS that is assigned to a configured endpoint, as specified by the NCS MIB table pktcNcsEndPointConfigTable. This list of CMSs MUST include only CMSs that are listed in the pktcMtaDevCmsTable.
2. For each CMS in the above list, the MTA MUST attempt to establish IPsec Security Associations as follows:
   a. Find the corresponding CMS table entry.
   b. If the MTA doesn't already possess a valid ticket for the specified CMS, use the pktcMtaDevCmsKerbRealmName parameter in the CMS table entry to index into pktcMtaDevRealmTable. Then, using the parameters associated with that realm perform steps SEC-1 through SEC-6 and optionally SEC-7 and SEC-8 in order to obtain the desired CMS ticket.
   c. Perform IPsec key management according to flows SEC-9 and SEC-10. This step MAY occur at any time after step b. above, but it must occur before any signaling messages are exchanged with that CMS.

   The CMS table entry contains various timing parameters used in steps SEC-9 and SEC-10. In the case of time outs or other errors, the MTA may retry using the timing parameters specified in the CMS table entry.

   The above steps MUST also apply when an additional MTA endpoint is activated (See [4]) or when an endpoint is configured (via SNMP sets) for a new CMS in the NCS MIB (see [26] and [47]).
3. Any time before an MTA endpoint sends a signaling message to a particular CMS, it MUST ensure that the respective Security Association is present. If the MTA is unable to establish IPsec SAs with a CMS that is associated with a configured endpoint (by the NCS MIB), it MUST set the NCS MIB variable pktcNcsEndPntStatusError to noSecurityAssociation (2).

After the initial establishment of the IPsec Security Associations for CMSs, the MTA MIB is utilized in subsequent key management as follows:

When the MTA receives a Wake Up message, it MUST respond with an AP Request when the corresponding CMS FQDN is found in the pktcMtaDevCmsTable and MUST NOT respond otherwise.

Note that establishment of IPsec Security Associations due to a Wake Up does not result in any call signaling traffic between the MTA and the CMS.

7.1.1.2.5.1 Call Flows SEC-5,6: Get a Kerberos Ticket for the CMS

The MTA uses PKINIT protocol to get a Kerberos Ticket for the specified CMS (see Section 6.4.3). After the KDC receives a ticket request, it retrieves the MTA FQDN from the provisioning server so that it can verify the request before replying with a ticket. The Telephony KDC issues the Kerberos Ticket for a group of one or more CMSs uniquely identified with the pair (Kerberos Realm, CMS Principal Name).

In the event that different MTA ports are configured for a different group of CMSs, the MTA MUST obtain multiple Kerberos Tickets by repeating these call flows for each CMS. Note that there is no requirement that the MTA obtain all the tickets from a single KDC.

7.1.1.2.5.2 Call Flows SEC-7,8,9: Establish IPsec SAs with the CMS

The MTA uses the Kerberos Ticket to establish a pair of simplex IPsec Security Associations with the given CMS. In the event that different MTA ports are configured with different CMS FQDN names, multiple pairs of SAs will be established (one set for each CMS).

When a single Kerberos ticket is issued for clustered Call Agents, it is used to establish more than one pair of IPsec SAs.

A CMS FQDN MAY translate into a list of multiple IP addresses, as would be the case with the NCS clustered Call Agents. In those cases, the MTA MUST initiate Kerberized key management with one of the IP addresses returned by the DNS Server. The MTA MAY also establish SAs with the additional CMS IP addresses.

Additional IPsec SAs with the other IP addresses MAY be established later, as needed (e.g., the current CMS IP address does not respond).

7.1.1.3 Key Management

7.1.1.3.1 MTA – SNMP Manager

Key Management for the MTA-Provisioning SNMPv3 user MUST use the Kerberized key management protocol as it is specified in Section 6.5.4. The MTA and the Provisioning Server MUST support this key management protocol. Additional SNMPv3 users MAY be created with the standard SNMPv3 cloning method [28] or with the same Kerberized key management protocol.

In order to perform Kerberized key management, the MTA must first locate the KDC. It retrieves the provisioning realm name from DHCP and then uses a DNS SRV record lookup to find the KDC FQDN(s) based on the realm name (see Section 6.4.5.1). When there is more than one KDC (DNS SRV record) found, DNS assigns a priority (and possibly a weighting) to each one. The MTA will choose a KDC based on the DNS priority and weight labeling and will go through the list until it finds a KDC that is able to respond.

7.1.1.3.2 MTA – TFTP Server

The optional encryption key for the MTA configuration file is passed to the MTA with an SNMP Set command (by the Provisioning Server) shown in the provisioning flow MTA-19. SNMPv3 security is utilized to provide message integrity and privacy. In the event that SNMPv3 privacy is not enabled, the MTA configuration file MUST NOT be encrypted and the file encryption key MUST NOT be passed to the MTA.

The encryption algorithm used to encrypt the file MUST be the same as the one used for SNMPv3 privacy. The same file encryption key MAY be re-used on the same configuration file while the MTA configuration file contents are unchanged. However, if the MTA configuration file changes or if a different encryption algorithm is selected for SNMPv3 privacy, the Provisioning Server MUST generate a new encryption key, MUST re-encrypt the configuration file and MUST update the TFTP Server with the re-encrypted file.

7.1.1.4 MTA Embedded Keys

The MTA device MUST be manufactured with a public/private RSA key pair and an X.509 device certificate that MUST be different from the BPI+ device certificate.

7.1.1.5 Summary Security Profile Matrix – Device Provisioning

The following matrix applies only to the Secure Provisioning Flow and SNMPv3.

Table 22. Security Profile Matrix – MTA Device Provisioning

	SNMP	TFTP (MTA – TFTP server)
authentication	Yes	Yes: authentication of source of configuration data.
access control	Yes: write access to MTA configuration is limited to authorized SNMP users. Read access can also be limited to the valid users when confidentiality is enabled.	Yes: write access to the TFTP server must be limited to the Provisioning Server but is out of scope for PacketCable. Read access can be optionally indirectly enabled when the MTA configuration file is encrypted.
integrity	Yes	Yes
confidentiality	Optional	Optional (of MTA configuration information during the TFTP-get)
non-repudiation	No	No
security mechanisms	SNMPv3 authentication and privacy. Kerberized key management protocol defined by PacketCable.	Hash of the MTA configuration file is sent to the MTA over SNMPv3, providing file authentication. When the file is encrypted, the key is also sent to the MTA over SNMPv3 (with SNMPv3 encryption turned on).

7.1.2 Subscriber Enrollment

The subscriber enrollment process establishes a permanent customer billing account that uniquely identifies the MTA to the CMS via the endpoint ID, which contains the MTA's FQDN. The billing account is also used to identify the services subscribed to by the customer for the MTA.

Subscriber enrollment MAY occur in-band or out-of-band. The actual specification of the subscriber enrollment process is out of scope for PacketCable and may be different for each Service Provider. The device provisioning procedure described in the previous section allows the MTA to establish IPsec Security Associations with one or more Call Agents, regardless of whether or not the corresponding subscriber had been enrolled.

As a result, when subscriber enrollment is performed in-band, a communication to a CSR (or to an automated subscriber enrollment system) is protected using the same security mechanisms that are used to secure all other voice communication.

During each communication setup (protected with IPsec ESP), the CMS MUST check the identity of an MTA against its authorization database to validate which voice communications services are permitted. If that MTA does not yet correspond to an enrolled subscriber, it will be restricted to permitting a customer to contact the service provider to establish service ("customer enrollment"). Some additional services, such as communications with emergency response organizations (e.g., 911), may also be permitted in this case. Since in-band customer enrollment is based on standard security provided for call signaling and media streams, no further details are provided in this section. Refer to Section 7.6 and to Section 6.6 on media streams.

7.2 Quality of Service (QoS) Signaling

7.2.1 Dynamic Quality of Service (DQoS)

7.2.1.1 Reference architecture for embedded MTAs

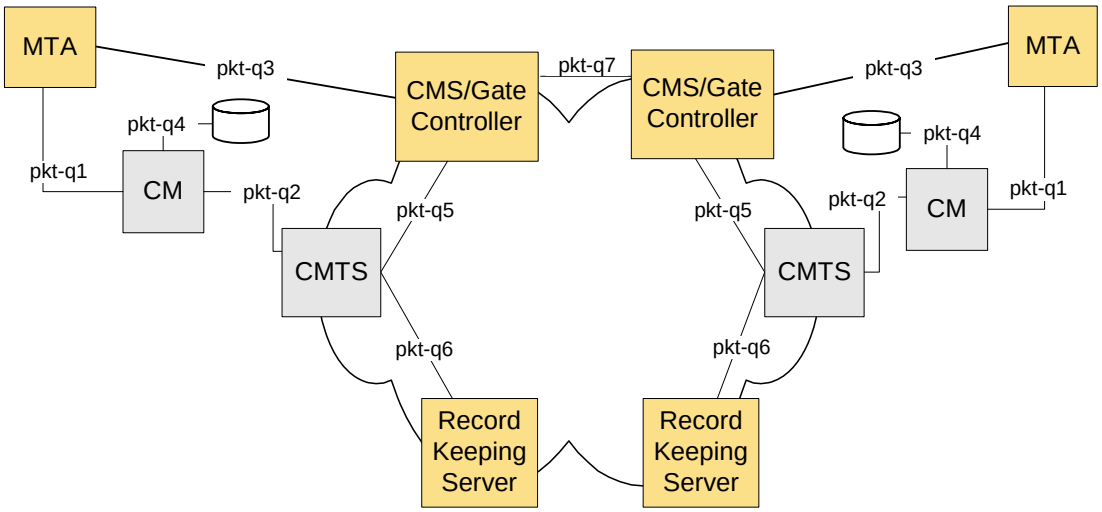


Figure 11. QoS Signaling Interfaces in PacketCable Network

7.2.1.2 Security Services

7.2.1.2.1 CM-CMTS DOCSIS 1.1 QoS Messages

Refer to the DOCSIS 1.1 RFI spec [8].

7.2.1.2.2 Gate Controller – CMTS COPS Messages

Authentication, Access Control and Message Integrity: required to prevent QoS theft and denial-of-service attacks.

Confidentiality: required to keep customer information private.

7.2.1.3 Cryptographic Mechanisms

7.2.1.3.1 CM-CMTS DOCSIS 1.1 QoS Messages

The DOCSIS 1.1 QoS messages are specified in the DOCSIS 1.1 RFI spec [8].

7.2.1.3.1.1 QoS Service Flow

A Service Flow is a DOCSIS MAC-layer transport service that provides unidirectional transport of packets either to upstream packets transmitted by the CM or to downstream packets transmitted by the CMTS. A service flow is characterized by a set of QoS Parameters such as latency, jitter, and throughput assurances. In order to standardize operation between the CM and CMTS, these attributes include details of how the CM requests mini-slots and the expected behavior of the CMTS upstream scheduler.

DOCSIS defines a Classifier, which consists of some packet matching criteria (IP source address, for example), a Classifier priority, and a reference to a service flow. If a packet matches the specified packet matching criteria, it is then delivered on the referenced service flow.

Downstream Classifiers are applied by the CMTS to packets it is transmitting, and Upstream Classifiers are applied at the CM and may be applied at the CMTS to police the classification of upstream packets.

The network can be vulnerable to IP packet attacks; i.e., attacks stemming from an attacker using another MTA's IP source address and flooding the network with the packets intended for another MTA's destination address. A CMTS controlling downstream service flows will limit an MTA's downstream bandwidth according to QoS allocations. If the CMTS is flooded from the backbone network with extra packets intended for one of its MTAs, packets for that MTA may be dropped to limit the downstream packet rate to its QoS allocation. The influx of the attacker’s packets may result in the dropping of good packets intended for the destination MTA.

To thwart this type of network attack, access to the backbone network should be controlled at the entry point. This can be accomplished using a variety of QoS Classifiers, but is most effective when the packet source is verified by its source IP address. This will limit the ability of a rogue source to flood the network with unauthorized IP packets.

CMTSs SHOULD use classifiers to police upstream packets (including verifying source IP addresses) arriving over the HFC access network.

For more information regarding the use of packet Classifiers, refer to the DOCSIS 1.1 RFI spec [8].

7.2.1.3.2 Gate Controller – CMTS COPS Messages

To download a QoS policy for a particular communications connection, the Gate Controller function in the CMS must send COPS messages to the CMTS. These COPS messages MUST be both authenticated and encrypted with IPsec ESP. Refer to Section 6.1.2 on the details of how IPsec ESP is used within PacketCable and for the list of available ciphersuites.

7.2.1.4 Key Management

7.2.1.4.1 Gate Controller – CMTS COPS Messages

Key management for this COPS interface is either IKE or Kerberos. Implementations MUST support IKE with pre-shared keys. Implementations MAY support IKE with X.509 certificates and they MAY support Kerberos using symmetric keys. For more information on the PacketCable use of IKE, refer to Section 6.2.2. For more information on the PacketCable use of Kerberos with symmetric keys, refer to Sections 6.4.3 and 6.5.

When the Gate Controller detects a failure of all COPS connections associated with a particular outgoing IPsec SA, it MUST delete all associated SAs (IKE and IPsec SAs if IKE is used as the Key management protocol or only IPsec SAs if Kerberos is used as the Key management protocol).

Subsequently, every N times (1<= N <= 10) that the Gate Controller tries to recover the connection, the SAs MUST be removed.

7.2.1.4.2 Security Profile Matrix Summary

Table 23. Security Profile Matrix – DQoS

	COPS (CMTS-CMS)
Authentication	Yes
access control	Yes
Integrity	Yes
Confidentiality	yes
non-repudiation	No
security mechanisms	IPsec with encryption and message integrity IKE or Kerberos

7.3 Billing System Interfaces

7.3.1 Security Services

7.3.1.1 CMS-RKS Interface

Authentication, Access Control and Message Integrity: required to prevent service theft and denial-of-service attacks. Want to insure that the billing events reported to the RKS are not falsified.

Confidentiality: required to protect subscriber information and communication patterns.

7.3.1.2 CMTS-RKS Interface

Authentication, Access Control and Message Integrity: required to prevent service theft and denial-of-service attacks. Want to insure that the billing events reported to the RKS are not falsified.

Confidentiality: required to protect subscriber information and communication patterns. Also, effective QoS information and network performance is kept secret from competitors.

7.3.1.3 MGC – RKS Interface

Authentication, Access Control and Message Integrity: required to prevent service theft and denial-of-service attacks. Want to insure that the billing events reported to the RKS are not falsified.

Confidentiality: required to protect subscriber information and communication patterns.

7.3.2 Cryptographic Mechanisms

Both message integrity and privacy MUST be provided by IPsec ESP, using any of the ciphersuites that are listed in Section 6.1.2.

RADIUS itself defines MD5-based keyed MAC for message integrity at the application layer. And, there does not appear to be a way to turn off this additional integrity check at the application layer. For PacketCable, the key for this RADIUS MAC MUST always be hardcoded to the value of 16 ASCII 0s. This in effect turns the RADIUS keyed MAC into an MD5 hash that can be used to protect against transmission errors but does not provide message integrity. No key management is needed for RADIUS MACs.

Billing event messages contain an 8-octet Element ID of the CMS, CMTS or the MGC. The RKS MUST verify each billing event by ensuring that the specified Element ID correctly corresponds to the IP address. This check is done via a lookup into a map of IP addresses to Element IDs. Refer to Section 7.3.3 on how

this map is maintained. A combined element (such as a combined CMS/MGC) MAY use the same IP address and Security Association to convey Event Messages from both elements. Additionally, both elements may use the same Element ID. Refer to Section 7.3.3.1 for information on how to maintain a map of multiple elements and Element IDs.

7.3.2.1 RADIUS Server Chaining

RADIUS servers may be chained. This means that when the local RADIUS server that is directly talking to the CMS or CMTS client is not able to process a message, it forwards it to the next server in the chain.

PacketCable specifies security mechanisms only on the links to the local RADIUS server. PacketCable also requires authentication, access control, message integrity and privacy on the interfaces between the chained RADIUS servers, but the corresponding specifications are outside of the scope of PacketCable.

Key Management (in the following section) applies to the local RADIUS Server/RKS only.

7.3.3 Key Management

7.3.3.1 CMS – RKS Interface

The CMS and the RKS MUST negotiate a shared secret (CMS-RKS Secret) using IKE or Kerberos with symmetric keys (implementations MUST support IKE with pre-shared keys; they MAY support IKE with X.509 certificates and they MAY support Kerberos using symmetric keys). For more information on the PacketCable use of IKE, refer to Section 6.2.2. For more information on the PacketCable use of Kerberos with symmetric keys, refer to Sections 6.4.3 and 6.5.

The key management protocol MUST run asynchronous to billing event generation, and will guarantee that there is always a valid, non-expired CMS-RKS Secret.

An RKS MUST maintain a mapping between an IP address and an Element ID for each host with which it has IPsec Security Associations. How this mapping is created depends on the IPsec key management protocol:

1. IKE with Pre-Shared Keys. One way to implement this mapping is to provide a local database of which Element ID(s) are associated with the source IP address.
2. IKE with Certificates. As specified in Section 8.2.3.4.3, a certificate of a server that sends billing event messages to an RKS contains its Element ID(s) in the CN attribute of the distinguished name. During IKE phase 1, the RKS MUST save a mapping between the IP address and its Element ID(s) that is contained in the certificate.
3. Kerberized Key Management. As specified in Section 6.4.5.5, a principal name of each server that reports billing event messages to the RKS includes its Element ID(s). After an RKS receives and validates an AP Request message, it MUST save a mapping between the IP address and its Element ID(s) that is contained in the principal name.

When an event message arrives at the RKS, the RKS MUST retrieve a source IP address based on the Element ID, using the mapping established during key management. The RKS MUST ensure that this address is the same as the source IP address in the IP packet header.

7.3.3.2 CMTS – RKS Interface

The CMTS and the RKS MUST negotiate a shared secret (CMTS-RKS Secret) using IKE or Kerberos (implementations MUST support IKE with pre-shared keys; they MAY support IKE with X.509 certificates and they MAY support Kerberos using symmetric keys). For more information on the PacketCable use of IKE, refer to Section 6.2.2. For more information on the PacketCable use of Kerberos with symmetric keys, refer to Sections 6.4.3 and 6.5.

The key management protocol MUST be running asynchronous to billing event generation, and will guarantee that there is always a valid, non-expired CMTS-RKS Secret.

An RKS maintains a mapping between an IP address and an Element ID for each host with which it has IPsec Security Associations, as specified in Section 7.3.3.1. This includes the CMTS.

When a billing event arrives at the RKS, the RKS MUST retrieve a source IP address based on the Element ID, using the mapping established during key management. The RKS MUST ensure that this address is the same as the source IP address in the IP packet header

7.3.3.3 MGC – RKS Interface

The MGC and the RKS MUST negotiate a shared secret (MGC-RKS Secret) using IKE or Kerberos (implementations MUST support IKE with pre-shared keys; they MAY support IKE with X.509 certificates and they MAY support Kerberos using pre-shared keys). For more information on the PacketCable use of IKE, refer to Section 6.2.2. For more information on the PacketCable use of Kerberos with symmetric keys, refer to Sections 6.4.3 and 6.5.

The key management protocol MUST be running asynchronous to billing event generation, and will guarantee that there is always a valid, non-expired MGC-RKS Secret.

An RKS maintains a mapping between an IP address and an Element ID for each host with which it has IPsec Security Associations, as specified in Section 7.3.3.1. This includes the MGC.

When an event message arrives at the RKS, the RKS MUST retrieve a source IP address based on the Element ID, based on the mapping established during key management. The RKS MUST ensure that this address is the same as the source IP address in the IP packet header.

7.3.4 Billing System Summary Security Profile Matrix

Table 24. Security Profile Matrix – RADIUS

	RADIUS Accounting (CMS - RADIUS Server/RKS)	**RADIUS Accounting (CMTS – RADIUS Server/RKS)**	**RADIUS Accounting (MGC – RADIUS Server/RKS)**
authentication	yes	yes	yes
access control	yes	yes	yes
integrity	yes	yes	yes
confidentiality	yes	yes	yes
non-repudiation	no	no	no
security mechanisms	IPsec ESP with encryption and message integrity enabled. key management using IKE or Kerberos	IPsec ESP with encryption and message integrity enabled key management using IKE or Kerberos	IPsec ESP with encryption and message integrity enabled key management using IKE or Kerberos

7.4 Call Signaling

7.4.1 Network Call Signaling (NCS)

7.4.1.1 Reference Architecture

The following diagram shows the network components and the various interfaces to be discussed in this section.

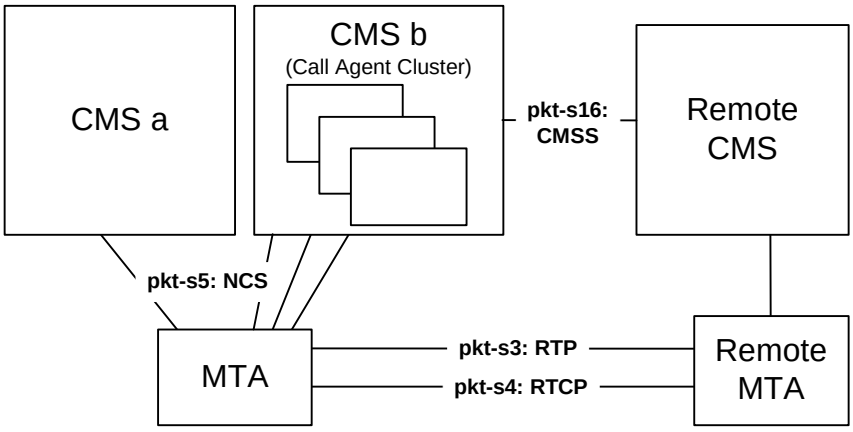


Figure 12. NCS Reference Architecture

Figure 12 shows a CMS containing a cluster of Call Agents, which are identifiable by one CMS FQDN. It also shows, even though this is not a likely scenario in early deployments, that different CMSs could potentially manage different endpoints in a single MTA.

The security aspects of interfaces pkt-s3 and pkt-s4 (RTP bearer channel and RTCP) are described in Section 6.6 of this document. The protocol interface pkt-s16 (CMS to CMS) is SIP with PacketCable extensions, as specified in [32].

When a call is made between two endpoints in different zones, the call signaling has to traverse the path between two different CMSs. The signaling protocol between CMSs is SIP with PacketCable specific extensions. See [32] for more details. Initially, the initiating CMS may not have a direct signaling path to a terminating CMS. The call routing table of the initiating CMS may point it to an intermediate SIP proxy. That SIP proxy, in turn, may point to another SIP proxy. In general, we make no assumptions about the number of SIP proxies in the signaling path between the CMSs. Once the two CMSs have discovered each other's location, they have the option to continue SIP signaling directly between each other. The SIP proxies that route traffic between Domains are called Exterior Border Proxies (EBPs). EBPs enforce access control on all signaling messages routed between domains. They also provide application level security on sensitive information contained within SIP messages. While not depicted in Figure 12, CMSS may also be used between a CMS and an MGC.

As SIP proxies and CMSs may be in different PacketCable domains (and consequently different trust domains), there must be a signaling path and trust relationship between two domains, before any direct SIP signaling can take place. PacketCable Server certificates are used for TLS mutual authentication in CMSS and provide the trust infrastructure for SIP signaling. A CMS or EBP, may be configured to only trust specific Service Provider CA certificates and/or FQDNs (i.e. access list) of external CMSs and EBPs. Generally, trust between different Service Provider domains is be provided by the EBPs.

7.4.1.2 Security Services

The same set of requirements applies to both CMS-MTA and CMS-CMS signaling interfaces.

Authentication: signaling messages should be authenticated, in order to prevent a third party masquerading as either an authorized MTA, CMS, MGC, or SIP Proxy.

Confidentiality: NCS messages carry dialed numbers and other customer information, which must not be disclosed to a third party. Thus confidentiality of signaling messages should be required. The signaling messages carry media stream keying material that must be kept private on each signaling hop, and should also be kept private end-to-end between the initiating and target CMSs, to avoid exposure at SIP signaling

proxies. There is no standard, well-supported mechanism to support end-to-end privacy of keying material, however, so only hop-by-hop confidentiality is supported in PacketCable.

Message integrity: should be assured in order to prevent tampering with signaling messages – e.g., changing the dialed numbers.

Access control: Services enabled by the NCS signaling should be made available only to authorized users – thus access control is required at the CMS.

7.4.1.3 Cryptographic Mechanisms

IPSec ESP MUST be used to secure the NCS signaling between the CMS and MTA. IPsec keys MUST be derived using the mechanism described in Section 6.5.3.1.

TLS MUST be used to secure the SIP signaling (CMSS) between CMSs and between CMSs and SIP proxies (EBPs).

The first SIP signaling roundtrip between the initiating and target CMSs may transit through any number of intermediate SIP signaling proxies. Since TLS is applied separately on each signaling hop, the contents of the SIP signaling message is decrypted and re-encrypted at each SIP signaling proxy. The full contents of the SIP signaling message, including media stream keying material, are available in the clear at each intermediate SIP signaling proxy.

7.4.1.3.1 MTA-CMS Interface

Each signaling message coming from the MTA and containing the MTA domain name (included in the NCS endpoint ID field) must be authenticated by the CMS. This domain name is an application-level NCS identifier that will be used by the Call Agent to associate the communication with a paying subscriber. In order to perform this authentication, the CMS MUST maintain an IP address to FQDN map for each MTA IP address that has a current SA. This map MUST be built during the key management process described in the following section and does not need to reside in permanent storage.

7.4.1.3.2 CMS-CMS, CMS-MGC, CMS-SIP Proxy and SIP Proxy – SIP Proxy Interfaces

When a CMS or MGC or a SIP Proxy receives a SIP signaling message, it SHOULD map the source IP address to the identity (FQDN) of the CMS or SIP Proxy and to the local policy associated with that FQDN. This lookup would utilize an IP address to FQDN map for all MGCs and SIP Proxies that have current TLS sessions with this host. This map is built during key management described in the following section and does not need to reside in permanent storage.

7.4.1.4 Key Management

7.4.1.4.1 MTA-CMS Key Management

The MTA MUST use Kerberos with PKINIT to obtain a CMS service ticket (see Section 6.4.3). The MTA SHOULD first obtain a TGT (Ticket Granting Ticket) via the AS Request/AS Reply exchange with the KDC (authenticated with PKINIT). In the case that the MTA obtained a TGT, it performs a TGS Request/TGS Reply exchange to obtain the CMS service ticket (see Section 6.4.4).

After the MTA has obtained a CMS ticket, it MUST execute a Kerberized key management protocol (that utilizes the CMS ticket) with the CMS to create SAs for the pkt-s10 interface. This Kerberized key management protocol is specified in Section 6.5. Section 6.5 also describes the mechanism to be deployed to handle timed-out IPsec keys and Kerberos tickets. The mechanism for transparently handling key switchover from one key lifetime to another key lifetime is also defined.

The key distribution and timeout mechanism is not linked to any specific NCS message. Rather, the MTA will obtain the Kerberos ticket from the KDC when started and will refresh it based on the timeout parameter. Similarly, the MTA will obtain the sub-key (and thus IPsec ESP keys) based on the IPsec timeout parameters. In addition, when the IPsec ESP keys are timed out and the MTA needs to transmit data to the CMS, it will perform key management with the CMS and obtain the new keys. It is also possible

for the IPsec SAs to expire at the CMS while it has data to send to the MTA. In this case, Section 6.5.3.5.3 describes the technique for the CMS to initiate key management and establish new Security Associations.

7.4.1.4.1.1 Call Agent Clustering

At the time that the CMS receives a Kerberos ticket for establishing an IPsec SA, it MUST extract the MTA FQDN from the MTA principal name in the ticket and map it to the IP address. This map is later used to authenticate the MTA endpoint ID in the NCS signaling messages.

In the case a CMS, or an application server, is constructed as a cluster of Call Agents with different IP addresses, all Call Agents should share the same service key for decrypting a Kerberos ticket. Thus the MTA will need to execute single PKINIT Request/Reply sequence with the KDC and multiple AP Request/Reply sequence for each Call Agent in the cluster. The Kerberos messages are specified in Section 6.4.4.

Optimized key management is specified for the case when in the middle of a communication, a clustered Call Agent sends a message to an MTA from a new IP address, where it doesn't yet have a IPsec SA with that MTA (see Section 6.5.2.1).

In this optimized approach, the CMS sends a Rekey message instead of the Wake Up. This Rekey message is authenticated with a SHA-1 HMAC, using a Server Authentication Key, derived from a session key used to encrypt the last AP Reply sent from the same CMS (or another CMS with the same Kerberos Principal Name).

Additionally, the Rekey message includes IPsec parameters, to avoid the need for the AP Reply message. The MTA responds with a different version of the AP Request that includes the MTA-CMS Secret, normally sent by the CMS in the AP Reply. As a result, after the MTA responds with the AP Request, a new IPsec SA can be established with no further messages. The total price for establishing a new SA with this optimized approach is a single roundtrip time. This is illustrated in the following figure:

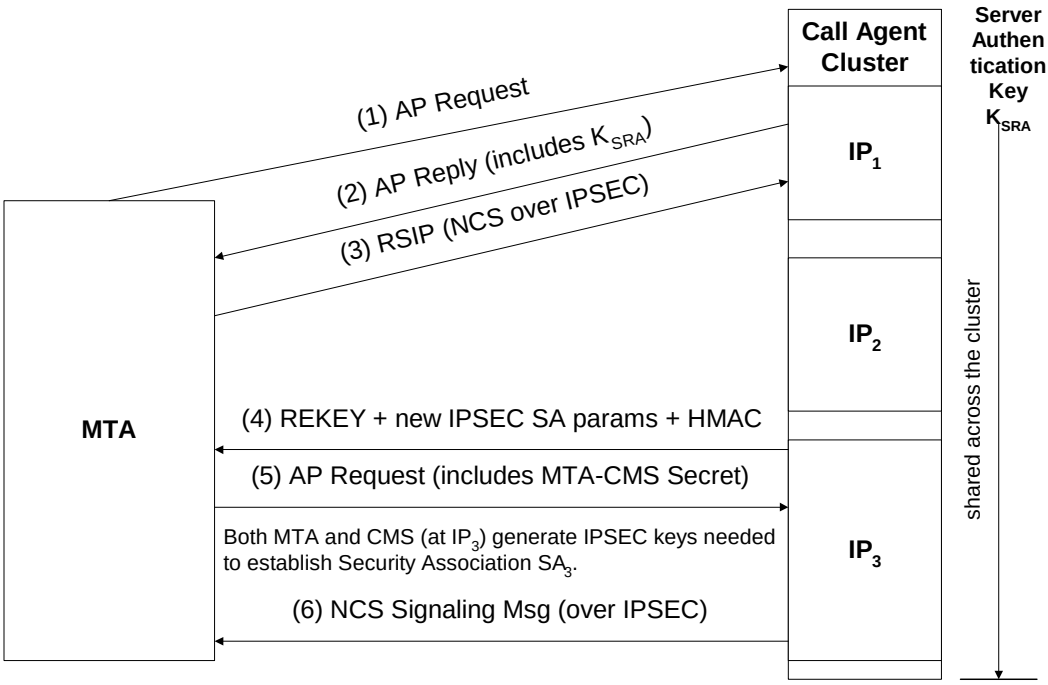


Figure 13. Key Management for NCS Clusters

In this figure, an NCS clustered Call Agent suddenly decides to send an NCS message from a new IP address that didn't previously have any SA established with that MTA.

The first Security Association SA_1 with CMS at IP_1 was established with a basic AP Request / AP Reply exchange. HMAC key K_{SRA} for authenticating Rekey message from the CMS was derived from the session key used to encrypt the AP Reply.

When a new SA_3 needs to be established between the MTA and CMS at IP_3, the key management is as follows:

(4) The CMS at IP_3 sends a REKEY message, similar in functionality to the Wake Up message, but with a significantly different content. It contains:

- IPsec parameters (also found in the AP Reply): SPI, selected ciphersuite, SA lifetime, grace period, and re-establish flag. The purpose of adding these IPsec parameters to REKEY is to eliminate the need for the subsequent AP Reply message.
- SHA-1 HMAC using K_{SRA}

(5) AP Request that includes the MTA-CMS secret, normally sent in the AP Reply message. This is a legal Kerberos mode, where the key is contained in the AP Request and AP Reply is not used at all.

For more details, refer to Section 6.5.3.

7.4.1.4.1.2 MTA Controlled by Multiple CMSs

In the case a single MTA is controlled by multiple CMSs and each CMS is associated with a different Kerberos realm, the MTA will need to execute multiple PKINIT Request/Reply exchanges with the KDC, one for each realm, optionally followed by a TGS Request/Reply exchanges. Then, an MTA would execute multiple AP Request/Reply exchanges in order to create the Security Associations with the individual CMSs.

7.4.1.4.1.3 Transferring from one CMS to another via NCS signaling

When control of an MTA endpoint is transferred from one CMS to another via NCS signaling, the following steps are taken:

1. The new CMS might not have been included in the CMS table. In that case, the corresponding table entry MUST be locally created. Refer to Section 7.1.1.2.5 for instructions on how to create the new CMS table entry.
2. If the MTA doesn't already have IPsec SAs established with this CMS (e.g., via an earlier Wake Up), it MUST attempt to establish them at this time.
3. If the MTA now possesses valid IPsec Security Associations with the new CMS, the NCS signaling software is notified and the Security Association can be utilized. Further signaling traffic for this affected endpoint related to the prior CMS Security Association MUST NOT be sent.

7.4.1.4.2 CMS-CMS, CMS-MGC, CMS-SIP Proxy, SIP Proxy-SIP Proxy Key Management

When a CMS, MGC, or a SIP Proxy has data to send to another CMS, MGC, or SIP Proxy and doesn't already have a TLS Session with that host, it MUST first establish a TLS session with the other CMS, MGC, or SIP Proxy (see Section 6.9).

A CMS or a SIP Proxy SHOULD create TLS sessions ahead of time (before they are needed) whenever possible and maintain persistent connections.

7.4.1.4.2.1 Example of Inter-Domain Call Setup with TLS Sessions

The following example diagram, depicts a typical SIP signaling flow for an inter-domain call setup in which EBPs are used (refer to [32] for further details on CMSS call flows). It illustrates several points in the end-end call setup where different TLS sessions and TCP connections may be required and also

emphasizes the importance of connection persistence and re-use to minimize TCP connection and TLS session establishment during the call setup (i.e. in order to minimize performance impacts and call setup delays). It should be noted that in a peering relationship between two SIP User Agents (i.e. CMSs and/or EBPs) often results in two TCP connections, one for SIP transactions initiated in each direction. This is due to the fact most TCP connections are initiated using ephemeral source ports and SIP transactions are initiated by sending SIP requests to a User Agent's well-known SIP port. As for securing each TCP connection with TLS, TLS clients typically cache TLS sessions based on specific remote IP address and port pairs, therefore it is unlikely TLS session caching using a common TLS master key can be used for both of the TLS sessions.

The inter-domain signaling flow begins with CMS "A" sending an INVITE to EBP "A" (CMS "A" is initiating a SIP INVITE transaction and will signal the well-known SIP port on EBP "A"). A TLS session is required and may need to be established for the TCP connection if one does not already exist (which can be re-used) for this new transaction. Similarly, a TLS session is required for each hop in this INVITE transaction, and may require a TLS session to be established between EBP "A" and EBP "B", and also between EBP "B" and CMS "B".

The 183 (Session Progress) response from CMS "B" is routed back through the EBPs to CMS "A" using the previously established TLS sessions. Once CMS "A" receives this 183 response, it sends a PRACK (Provisional ACK) directly to CMS "B". This PRACK is a SIP request which initiates a new transaction, and requires a TLS session be established with CMS "B"'s well-known SIP port. CMS "B" sends a 200 OK response back to CMS "A" (using the same TLS session and TCP connection) as a the final response to this PRACK transaction. Upon receiving a response to its initial INVITE, CMS "A" will also send an UPDATE request to CMS "B" over the previously established TLS session, to indicate resource reservation has been completed. CMS "B" will respond with a 200 OK, completing this UPDATE transaction.

Upon receiving the UPDATE, CMS "B" reserves any necessary resource and sends back a 180 (Ringing) provisional response to CMS "A" over the previously established TLS session. This provisional response will also initiate PRACK / 200 OK transaction between the two CMSs, over the same TLS session.

Once the terminating end answers the call, CMS "B" sends the 200 OK final response to the INVITE. However, this response is sent back via the EBPs using the same TLS sessions used for the INVITE. CMS "A" will acknowledge receipt of this 200 OK response by sending an ACK (a SIP request) directly to CMS "B". This ACK is sent using the previously established TLS session used for the first PRACK.

Once the call is established, the example illustrates the case where the terminating end goes on hook. CMS "B" initiates a BYE / 200 OK transaction by sending a BYE (SIP request) to CMS "A". As this BYE request is sent to the well-known SIP signaling port of CMS "A", it is very likely CMS "B" will need to use a different TCP connection and TLS session than the ones used for sending SIP requests from CMS "A" to CMS "B" (assuming CMS "A" is using an ephemeral port for its TCP connection to CMS "B").

As can be seen from this example, two TCP connections with two distinct TLS sessions may be required between two CMSs. It is important to support persistent and re-usable connections and TLS session caching in order to minimize impacts on CMS performance and call latency.

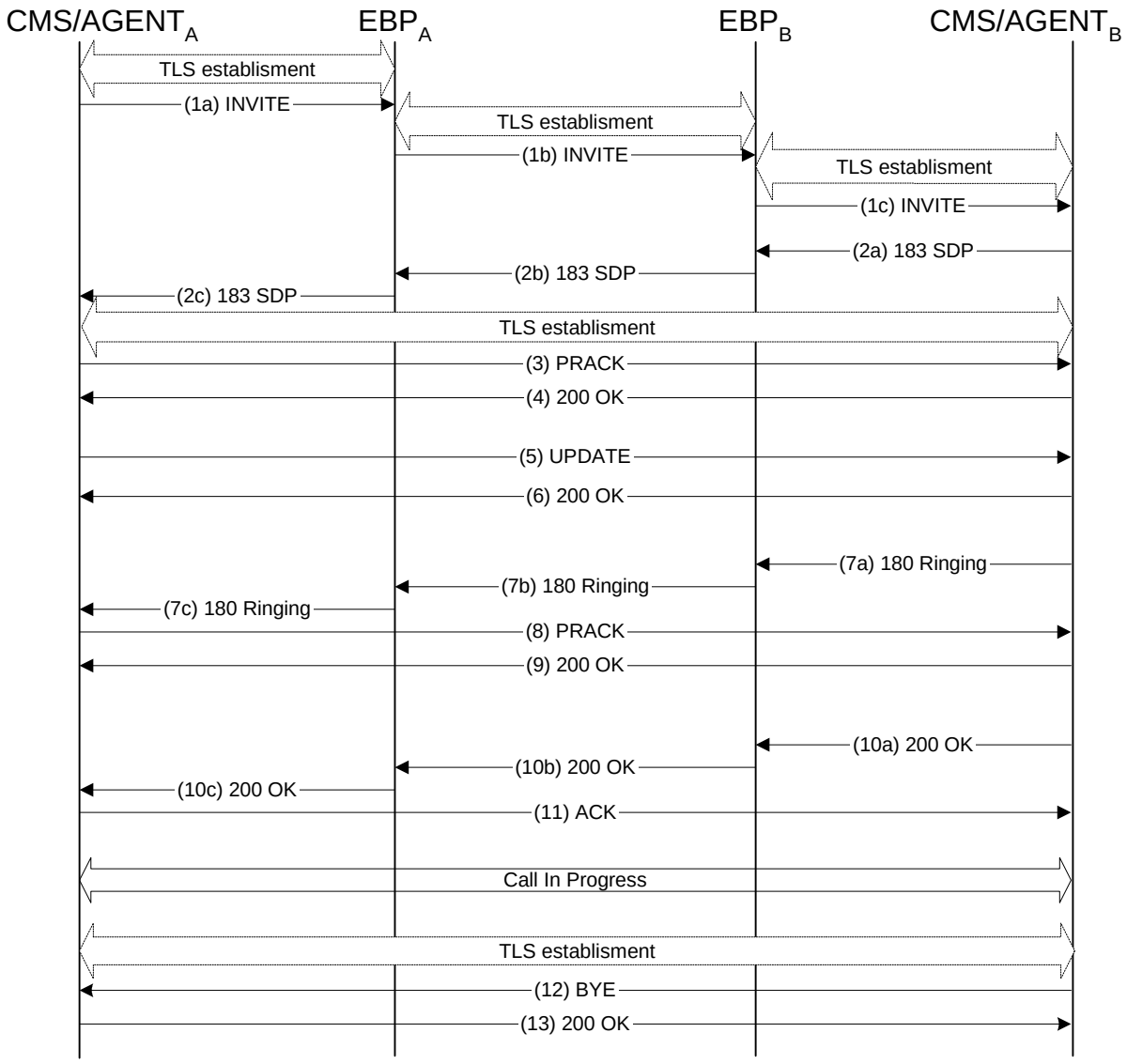


Figure 14. CMS – CMS Signaling Flow with Security

7.4.2 Call Signaling Security Profile Matrix

Table 25. Security Profile Matrix – Network Call Signaling

	MTA-CMS	CMS-CMS	CMS-SIP Proxy / SIP Proxy-SIP Proxy
authentication	optional	yes	yes
access control	optional	yes	yes
integrity	optional	yes	yes
confidentiality	optional	yes	yes
non-repudiation	no	no	no
security mechanisms	IPsec ESP with encryption and message integrity enabled	TLS with encryption and message integrity	TLS with encryption and message integrity

	MTA-CMS	CMS-CMS	CMS-SIP Proxy / SIP Proxy-SIP Proxy
	Authentication via Kerberos with PKINIT Kerberized key management defined by PacketCable Security may be disabled through the provisioning process.	Authentication via X.509 certificates (or symmetric keys when TLS session caching is used)	Authentication via X.509 certificates (or symmetric keys when TLS session caching is used)

7.5 PSTN Gateway Interface

7.5.1 Reference Architecture

A PacketCable PSTN Gateway consists of three functional components:

- a Media Gateway Controller (MGC) which may or may not be part of the CMS,
- a Media Gateway (MG), and
- a Signaling Gateway (SG).

These components are described in detail in [5].

7.5.1.1 Media Gateway Controller

The Media Gateway Controller (MGC) is the PSTN gateway's overall controller. The MGC receives and mediates call-signaling information between the PacketCable and the PSTN domains (from the SG), and it maintains and controls the overall state for all communications.

7.5.1.2 Media Gateway

Media Gateways (MG) provide the bearer connectivity between the PSTN and the PacketCable IP network.

7.5.1.3 Signaling Gateway

PacketCable provides support for SS7 signaling gateways. The SG contains the SG to MGC interface. Refer to [5] for more detail on signaling gateways.

The SS7 Signaling Gateway performs the following security-related functions:

- Isolates the SS7 network from the IP network. Guards the SS7 network from threats such as Information Leakage, integrity violation, denial-of-service, and illegitimate use.
- Provides mechanism for certain trusted entities ("TCAP Users") within the PacketCable network, such as Call Agents, to query external PSTN databases via TCAP messages sent over the SS7 network.

7.5.2 Security Services

7.5.2.1 MGC – MG Interface

Authentication: Both the MG and the MGC must be authenticated, in order to prevent a third party masquerading as either an authorized MGC or MG.

Access Control: MG resources should be made available only to authorized users – thus access control is required at the MG.

Integrity: must be assured in order to prevent tampering with the TGCP signaling messages – e.g., changing the dialed numbers.

Confidentiality: TGCP signaling messages carry dialed numbers and other customer information, which must not be disclosed to a third party. Thus confidentiality of the TGCP signaling messages is required.

7.5.3 Cryptographic Mechanisms

7.5.3.1 MGC – MG Interface

IPsec ESP MUST be used to both authenticate and encrypt the messages from MGC to MG and vice versa. Refer to Section 6.1.2 for details of how IPsec ESP is used within PacketCable and for the list of available ciphersuites.

7.5.4 Key Management

7.5.4.1 MGC – MG Interface

Key management for the MGC-MG interface is either IKE or Kerberos. Implementations MUST support IKE with pre-shared keys. Implementations MAY support IKE with X.509 certificates and they MAY support Kerberos using symmetric keys. For more information on the PacketCable use of IKE, refer to Section 6.2.2. For more information on the PacketCable use of Kerberos with symmetric keys, refer to Sections 6.4.3 and 6.5.

The key management protocol ensures that there is always a valid, non-expired MGC – MG secret.

7.5.5 MGC-MG Summary Security Profile Matrix

Table 26. Security Profile Matrix – TGCP

	TGCP (MG – MGC)
authentication	yes
access control	yes
integrity	yes
confidentiality	yes
non-repudiation	no
security mechanisms	IPsec IKE or Kerberos

7.6 Media Stream

This security specification allows for end-to-end ciphersuite negotiation, so that the communicating parties can choose their preferred encryption and authentication algorithms for the particular communication.

7.6.1 Security Services

7.6.1.1 RTP

Authentication: End-to-end authentication cannot be required, because the initiating party may want to keep their identity private. Optional end-to-end exchanges for both authentication and additional key negotiation are possible but are outside of the scope for PacketCable.

Encryption: The media stream between MTAs and/or MGs should be encrypted for privacy. Without encryption, the stream is vulnerable to eavesdropping at any point in the network.

Key Distribution via the CMS, a trusted third party, assures the MTA (or MG) that the communication was established through valid signaling procedures, and with a valid subscriber. All this guarantees confidentiality (but not authentication).

Message Integrity: It is desirable to provide each packet of the media stream with a message authentication code (MAC). A MAC ensures the receiver that the packet came from the legitimate sender and that it has not been tampered with en route. A MAC defends against a variety of potential known attacks, such as replay, clogging, etc. It also may defend against as-yet-undiscovered attacks. Typically, a MAC consists of 8 or more octets appended to the message being protected. In some situations, where data bandwidth is limited, a MAC of this size is inappropriate. As a tradeoff between security and bandwidth utilization, a short MAC consisting or 2 or 4 octets is specified and selectable as an option to protect media stream packets. Use of the MAC during an end-to-end connection is optional; whether it is used or not is decided during the end-to-end ciphersuite negotiation (see Section 7.6.2.3.1).

Low complexity: Media stream security must be easy to implement. Of particular concern is a PSTN gateway, which may have to apply security to thousands of media streams simultaneously. The encryption and MAC algorithms used with the PSTN gateway must be of low complexity so that it is practical to implement them on such a scale.

7.6.1.2 RTCP

Authentication: see the above section.

Encryption: within PacketCable, RTCP messages are not permitted to contain the identity of the RTCP termination endpoint. Snooping on RTCP messages, therefore, does not reveal any subscriber-specific information but may reveal network usage and reliability statistics. RTCP encryption is optional.

Message Integrity: RTCP signaling messages (e.g., BYE) can be manipulated to cause denial-of-service attacks and alteration of reception statistics. To prevent these attacks, message integrity should be used for RTCP.

7.6.2 Cryptographic Mechanisms

MTAs and MGs MUST have an ability to negotiate a particular encryption and authentication algorithm. If media security parameters are negotiated and RTP encryption is on (Transform ID is not RTP_ENCR_NULL), each media RTP packet MUST be encrypted for privacy. If RTP encryption is on, encryption MUST be applied to the RTP payload and MUST NOT be applied to the RTP header. Security MUST NOT be applied to RTP packets if the negotiated RTP ciphersuite is AUTH_NULL and RTP_ENCR_NULL. Each RTP packet MAY include an optional message authentication code (MAC). The MAC algorithm can also be negotiated. The MAC computation MUST span the packet's unencrypted header and encrypted payload. The receiver MUST perform the same computation as the sender and it MUST discard the received packet if the value in the MAC field does not match the computed value.

Keys for the encryption and MAC calculation MUST be derived from the End-End secret, which is exchanged between sending and receiving MTA as described in Section 7.6.2.3.1.

7.6.2.1 RTP Messages

Figure 15 shows the format of an encoded RTP packet. PacketCable MUST adhere to the RTP packet format as defined by RFC 1889 [10] and RFC 1890 [42] after being authenticated and decrypted (where the MAC bytes, if included, are stripped off as part of the authentication).

The packet's header consists of 12 or more octets, as described in [10]. The only field of the header that is relevant to the encoding process is the timestamp field.

The RTP header has the following format (RFC-1889):

0							1								2								3						
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
V=2		P	X	CC				M	PT							Sequence Number															
Timestamp																															
Synchronization Source (SSRC) Identifier																															
Contributing Source (CSRC) Identifier																															

Figure 15. RTP Packet Header Format

The first twelve octets are present in every RTP packet, while the list of CSRC identifiers is present only when inserted by a mixer.

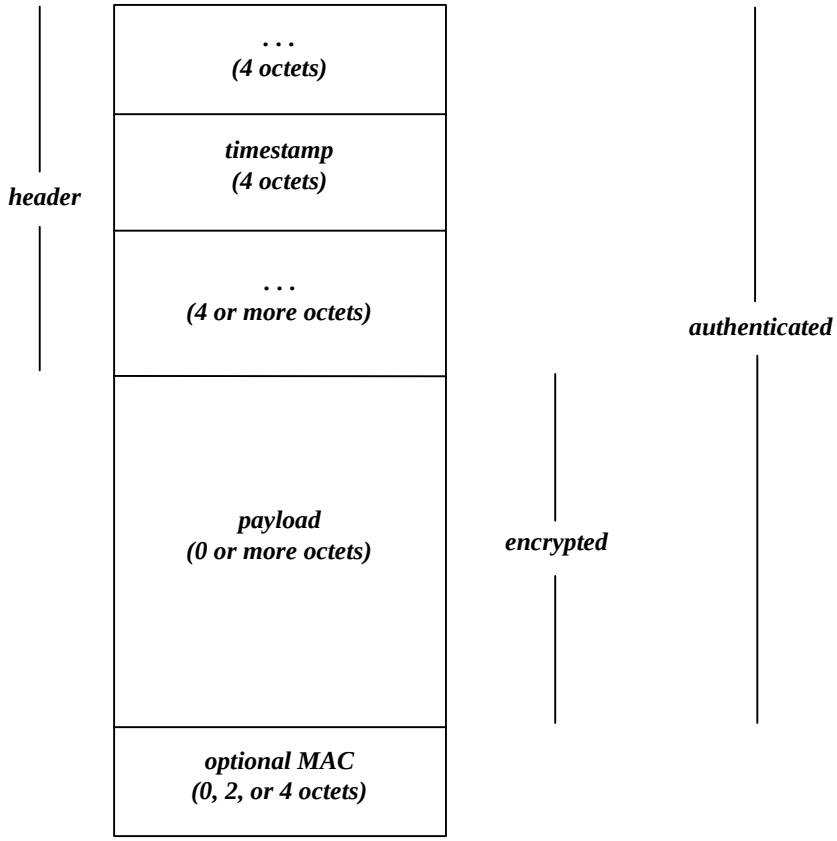


Figure 16. Format of Encoded RTP Packet

In PacketCable, an RTP packet will carry compressed audio from the sender’s voice codec, or it will carry a message describing one or more events such as a DTMF tone, trunk or line signaling, etc. For simplicity, the former is referred to as a "voice packet" and the latter as an "event packet."

A voice packet's payload consists of compressed audio from the sender's voice codec. The length of the payload is variable and depends on the voice codec as well as the number of codec frames carried by the packet.

An event packet's payload consists of a message describing the relevant event or events. The format of the message is outside the scope of this specification. The length of the payload is variable, but it will not exceed a known, maximum value.

For either type of packet, the payload MUST be encrypted. If the optional MAC is selected, the MAC field is appended to the end of the packet after the payload.

Parameters representing RTP packet characteristics are defined as follows:

- N_c, the number of octets in one frame of compressed audio. Each codec has a well-defined value of N_c. In the case of a codec that encodes silence using short frames, N_c refers to the number of octets in a nonsilent frame.
- N_u, the number of speech samples in one frame of uncompressed audio. The number of speech samples represented by a voice packet is an integral multiple of N_u.
- N_f, the frame number. The first frame of the sender's codec has a value of zero for N_f. Subsequent frames increment N_f by one. N_f increments regardless of whether a frame is actually transmitted or discarded as silent.
- M_f, the maximum number of frames per packet. M_f is determined by the codec's frame rate and by the sender's packetization rate. The packetization rate is specified during communications setup. For NCS signaling, it is a parameter in the LocalConnectionOptions – see [2].

  For example, suppose the speech sample rate is 8,000 samples/sec, the frame rate is 10 msec, the packetization rate is 30 msec, and the compressed audio rate is 16,000 bits/sec. Then $N_c = 20$, $N_u = 80$, $M_f = 3$, and N_f counts the sequence 0, 1, 2.

  N_e, the maximum number of bytes that might be sent within the duration of one codec frame. It is assumed that an event packet can have a payload as large as that of a voice packet, but no longer. In the case of a block cipher, the cryptographic keys do not change after midstream codec changes. When a codec change does not require a corresponding key change, the value of N_e MUST be calculated as follows:

  $$N_e = MAX \{ N_cK \} \text{ for } K = 1, \ldots N$$

  Where N1, N2, … NK are the different frame sizes for codecs that are supported by a particular endpoint.

  Otherwise, $N_e = N_c$, where N_c is the frame size for the current codec.
- N_m, the number of MAC octets. This value is 0, if the optional MAC is not selected; or 2 or 4, representing the MAC size if the optional MAC is selected.

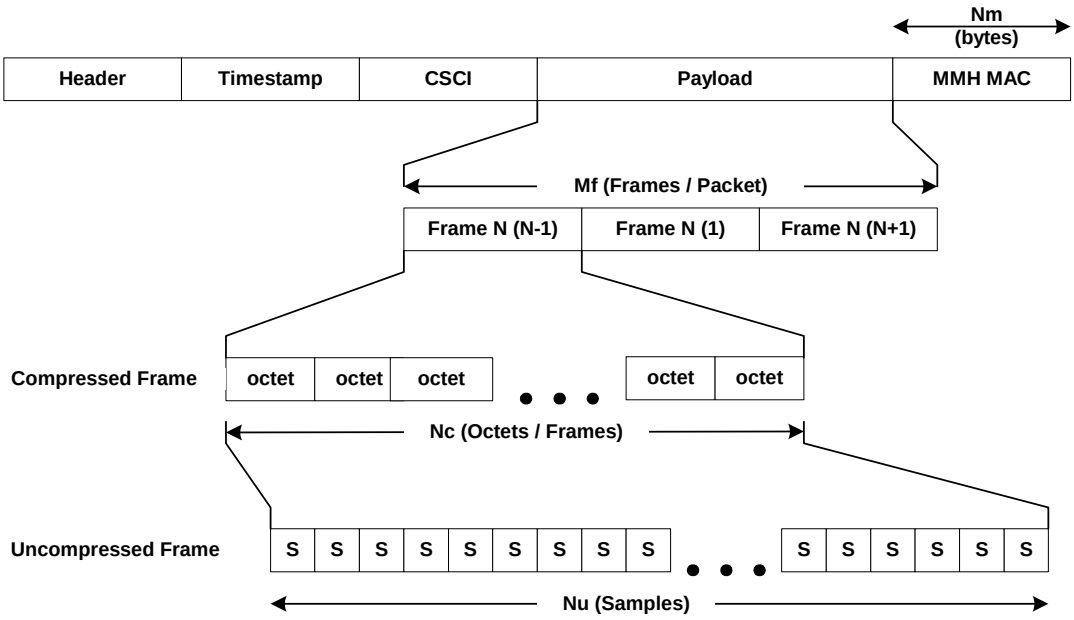


Figure 17. RTP Packet Profile Characteristics

7.6.2.1.1 RTP Timestamp

According to RFC1889, the timestamp field is a 32-bit value initially chosen at random. to PacketCable, the timestamp MUST increment according to the codec sampling frequency. The timestamp in the RTP header MUST reflect the sampling instant of the first octet in each RTP packet presented as offset from the initial random timestamp value. The timestamp field MAY be used by the receiver to synchronize its decryption process to the encryption process of the sender.

Based on the definition of the timestamp and the packet parameters described in the previous section, the timestamp MUST equate to the value: $((N_f * N_u) + (\text{RTP Initial Timestamp}))$ modulo 2^{32}, where N_f is the frame number of the first frame included in the packet.

7.6.2.1.2 Packet Encoding Requirements

Prior to encoding the packets of an RTP stream, the sending MTA MUST derive the keys and parameters from the End-End Secret it shares with the receiving MTA, as specified in Section 7.6.2.3.3.

An MTA MUST derive two distinct sets of these quantities, one set for processing outgoing packets and another set for processing incoming packets.

7.6.2.1.2.1 Encryption and MMH MAC Option

7.6.2.1.2.1.1 Deriving an MMH MAC Key

The MMH MAC Key size MUST be determined before generating the MMH MAC Key. The following algorithm specifies how to derive the MMH MAC Key when being used with block ciphers.

$$\text{MMH MAC key size} = (M_f * N_e) + N_h + N_m - 2 + P$$

Where: M_f is the maximum number of frames per packet; N_e is maximum number of octets in one frame of compressed audio; N_h is the maximum number of octets in the RTP header, as defined in Section 7.6.2.1; and N_m is the number of octets in the MAC. Therefore, $(M_f * N_e) + N_h$ represents the maximum size of an RTP packet, and $N_m - 2$ represents the additional two octets that are added to the key size when a four octet MMH MAC is used. (The key size is the same as the maximum RTP packet size when a two octet MMH

MAC is used.) P is 0 or 1, as needed to make the MMH MAC key size an even number so that it is a multiple of the word size (2 bytes) used in the MMH MAC algorithm.

The number of octets in the RTP header ranges from 12 to 72, inclusive, depending on the number of CSRC identifiers that are included [10]. An implementation MUST choose N_h at least as large as required to accommodate the maximum number of CSRC identifiers that may occur during a session. An implementation MUST set N_h to 72 if the maximum number of CSRC identifiers is otherwise unknown.

Since the key derivation procedure generates the MMH MAC key last (see Section 7.6.2.3.3.1), it is not necessary to generate a complete MMH MAC key at the start of the RTP session. Implementations MAY generate less than the full MMH MAC key and generate the rest later, as needed. For example, instead of using a value of N_e that reflects all possible codecs supported by an endpoint, an implementation might initially derive an MMH key of size $(M_f * N_c) + N_h + N_m - 2 + P$, where N_c is the frame size for the currently selected codec. Later, after a codec change that results in a larger value of N_c, additional bytes for the MMH key may be generated.

7.6.2.1.2.1.2 RTP Timestamp Wrap-around

Let us say that the initial RTP timestamp value is T_0. A timestamp wrap-around occurs when:

- an RTP packet with sequence number i has a timestamp value $2^{32} - \xi_1$ for $0 < \xi_1 <= \Delta T_{MAX}$, where ΔT_{MAX} is the maximum difference between two consecutive RTP timestamps.
- an RTP packet with a sequence number i+1 has a timestamp value ξ_2 for $0 <= \xi_2 < \Delta T_{MAX}$.

The wrap-around point is between the RTP packets i and i+1.

Each endpoint MUST keep a count N_{WRAP} of RTP timestamp wrap-arounds, with a range from 0 to 2^{16}-1 and initialized to zero at the start of the connection N_{WRAP} MUST be incremented by the sender right after the wrap-around point. N_{WRAP} MUST also be incremented by the receiver before it decrypts any RTP packets after the wrap-around point.

7.6.2.1.2.2 Block Cipher Encryption of RTP Packets

The AES Block Cipher must be supported for encryption of RTP packets. The following sections specify how to support any Block Cipher, including AES.

7.6.2.1.2.2.1 Block Termination

If an implementation supports block ciphers, residual block termination (RBT) MUST be used to terminate streams that end with less than a full block of data to encrypt (see Section 9.3).

7.6.2.1.2.2.2 Initialization Vector

An Initialization Vector (IV) is required when using a block cipher in CBC mode to encrypt RTP packet payloads. The size of an IV is the same as the block size for the particular block cipher. For example, the IV size for DESX and 3-DES is 64 bits, while for AES-CBC it is 128 bits. In order to calculate the IV each endpoint MUST keep track of N_{WRAP} - the count of timestamp wrap-arounds during this RTP session, see Section 7.6.2.1.2.1.2 The IV MUST be calculated new for each RTP packet as specified below:

1. Take the first N bits of the header, where N = min(cipher block size, RTP header size).
2. In the result of the previous step replace the first 16 bits of the header with the 16-bit value of N_{WRAP}, MSB first.
3. Pad the result of previous step with 0's on the right, so that the resulting bit string is equal in size to the cipher block size.
4. XOR the result of the previous step with the RTP Initialization Key (defined in Section 7.6.2.3.3.1). The size of the RTP Initialization Key is the same as the cipher block size.
5. Encrypt the result of the previous step using the same block cipher that is used to encrypt RTP packets, but in ECB mode. The result of this step is the Initialization Vector for this RTP packet.

7.6.2.1.2.2.3 MMH-MAC Pad Derivation When Using a Block Cipher

The MMH-MAC algorithm requires a one-time pad for each RTP packet. The MMH-MAC Pad MUST be derived by performing the MMH Function on the Block Cipher's IV. For a 2-byte MMH-MAC, use the MMH Function described in Section 9.7.1.1; for a 4-byte MMH-MAC, use the MMH Function described in Section 9.7.1.2.

The IV is calculated according to Section 7.6.2.1.2.2.2 for block ciphers that require an IV. Even if the block cipher does not require an IV, one MUST be derived according to Section 7.6.2.1.2.2.2 and used as the basis of the MMH-MAC Pad derivation.

A key is also required by the MMH digest function in order to calculate the pad. The MMH MAC key derived in Section 7.6.2.3.3.1 MUST be truncated according to Section 9.7.2.2 and MUST then be used as the key to the MMH digest. Accordingly, the MMH MAC key is truncated to:

$$<\text{size of IV}> + N_m - 2$$

Where <size of IV> is 16 bytes for AES, N_m is the size of the MMH MAC in bytes, as defined in Section 7.6.2.1, and $N_m - 2$ represents the additional two octets that are added to the key size when a four octet MMH MAC is used). (The truncated key size is the same as the IV size when a two octet MMH MAC is used.)

7.6.2.1.3 Packet Decoding Requirements

Prior to decoding the packets of an RTP stream, the receiving MTA MUST derive the keys and parameters from the End-End Secret it shares with the sending MTA, as specified in Section 7.6.2.3.3.

The derived quantities MUST match the corresponding quantities at the sending MTA.

7.6.2.1.3.1 Timestamp Tolerance Check

Before processing a received packet, the receiver SHOULD perform a sanity check on the timestamp value in the RTP header, consisting of items (1) and (2) below:

1. Beginning with the RTP timestamp in the first packet received from a sender, the receiver calculates an expected value for the timestamp of the sender's next RTP packet based on timestamps received in the sender's previous packets for the session.
2. The next packet is rejected without being processed if its timestamp value is outside a reasonable tolerance of the expected value. (Timestamps from rejected packets are not to be used to predict future packets). The tolerance value is defined to be:
    a. sufficiently tight to ensure that an invalid timestamp value cannot derail the receiver's state so much that it cannot quickly recover to decrypting valid packets.
    b. able to account for known differences in the expected and received timestamp values, such as might occur at call startup, codec switch over and due to sender/receiver clock drift.

If the timestamp value in the RTP headers from a sender never comes back within the acceptable range, the receiver discontinues the session.

At the receipt of each packet, the receiver adjusts its time relationship with the sender within the acceptable tolerance range of estimated values.

7.6.2.1.3.2 Packet Authentication

If authentication is used on an RTP packet stream, verification of the MAC MUST be the first step in the packet decoding process. When the timestamp tolerance check is performed, the MAC MAY be verified on packets with valid RTP timestamps immediately after the check is completed.

If the MAC does not verify, the packet MUST be rejected.

7.6.2.2 RTCP Messages

7.6.2.2.1 RTCP Format

RFC 1889 defines the packet format of RTCP messages.

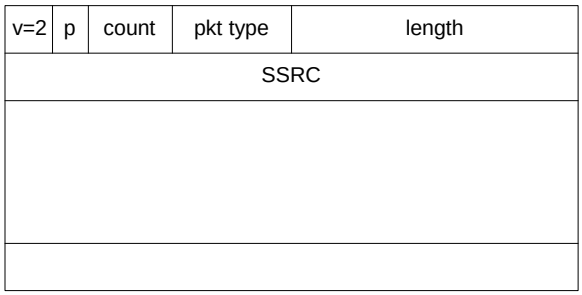


Figure 18. RTCP Packet Format

The RTCP packet type could be SR (sender reports), RR (receiver reports), SDES (source description), BYE (leaving conference), and APP (application specific function). The length varies depending on the message type, but generally around 40 bytes.

7.6.2.2.2 RTCP Encryption

RTCP messages MUST always be encrypted in their entirety when the negotiated encryption algorithm is a block cipher in CBC mode. RTCP messages MUST NOT be encrypted when the negotiated encryption algorithm is RTCP_ENCR_NULL. However, the encoded RTCP messages MUST still be formatted according to Section 7.6.2.2.2 when RTCP_ENCR_NULL is selected in conjunction with a non-NULL authentication algorithm (e.g., HMAC-SHA1-96 or HMAC-MD5-96). Security MUST NOT be applied to RTCP packets if the negotiated RTCP ciphersuite is RTCP_AUTH_NULL and RTCP_ENCR_NULL. After the message is encrypted, an additional header and MAC (Message Authentication Code) are added. The result packet has the format in the following diagram.

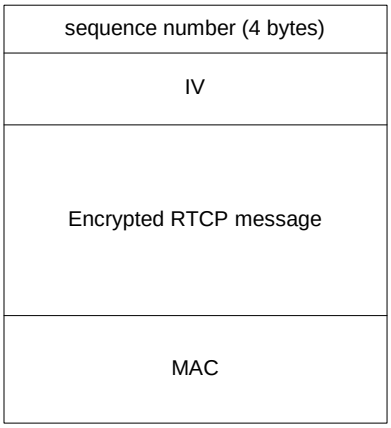


Figure 19. RTCP Encrypted Packet Format

The first 4 bytes MUST be the sequence number, MSB first. The initial sequence number for each direction of traffic MUST be 0. Afterwards, the sequence number for each direction MUST be incremented by 1. Generally, one RTCP message is sent every 5 seconds for each channel. Thus 32 bits for the sequence number field would be big enough for any connections without wrapping around.

The IV (Initialization Vector) MUST immediately follow the sequence number. The IV MUST be randomly generated by the sender for each RTCP message and the IV size MUST be the same as the block size for the selected block cipher. The Initialization Vector (IV) MUST NOT be included when RTCP_ENCR_NULL is used.

The original cleartext RTCP message encrypted in its entirety MUST immediately follow the IV. The MAC (Message Authentication Code) computed over the concatenation of the sequence number, IV and the encrypted message MUST follow the encrypted RTCP message. The size of the MAC is algorithm-dependent.

7.6.2.2.3 Sequence Numbers

The receiver of RTCP messages SHOULD keep a sliding window of the RTCP sequence numbers. The size of the sliding window W_{RTCP} depends on the reliability of the UDP transport and is locally configured at each endpoint. W_{RTCP} SHOULD be 32 or 64. The sliding window is most efficiently implemented with a bit mask and bit shift operations.

When the receiver is first ready to receive RTCP packets, the first sequence number in this window MUST be 0 and the last MUST be $W_{RTCP} - 1$. All sequence numbers within this window MUST be accepted the first time but MUST be rejected when they are repeated. All sequence numbers that are smaller than the "left" edge of the window MUST be rejected.

When an authenticated RTCP packet with a sequence number that is larger than the "right" edge of the window is received, that sequence number is accepted and the "right" edge of the window is replaced with this sequence number. The "left" edge of the window is updated in order to maintain the same window size.

When for a window $(S_{RIGHT} - W_{RTCP} + 1, S_{RIGHT})$, sequence number S_{NEW} is received and $S_{NEW} > S_{RIGHT}$, then the new window becomes:

$$(S_{NEW} - W_{RTCP} + 1, S_{NEW})$$

7.6.2.2.4 Block Termination

Residual block termination (RBT) MUST be used to terminate RTCP messages that end with less than a full block of data to encrypt (see Section 9.3).

7.6.2.2.5 RTCP Message Encoding

Each RTCP message MUST be encoded using the following procedure:

1. A random IV is generated.
2. The entire RTCP message is encrypted with the selected block cipher and the just generated IV.
3. The current sequence number, IV and the encrypted RTCP message are concatenated in that order.
4. The MAC is computed (using the selected MAC algorithm) over the result in c) and appended to the message.

7.6.2.2.6 RTCP Message Decoding

Each RTCP message MUST be decoded using the following procedure:

1. Regenerate the MAC code and compare to the received value. If the two don’t match, the message is dropped.
2. The sequence number is verified based on the sliding window approach specified in Section 7.6.2.2.3. If the sequence number is rejected, the message is dropped. The sliding window is also updated as specified in Section 7.6.2.2.3.
3. The RTCP message is decrypted with the shared encryption key and with the IV that is specified in the message header.

7.6.2.3 Key Management

The key management specified here for end-to-end communication is identical in the cases of the MTA–to-PSTN and MTA-to-MTA communications. In the case of the MTA-to-PSTN communications, one of the MTAs is replaced by a MG (Media Gateway).

The descriptions below refer to MTA-to-MTA communications only for simplicity. In this context, an MTA actually means a communication end point, which can be an MTA or a MG. In the case that the end point is a MG, it is controlled by an MGC instead of a CMS.

During call setup MTA_0 (the initiating MTA) and MTA_1 (the terminating MTA) exchange randomly generated keying material, carried inside the call signaling messages. Call signaling messages are themselves protected by IPsec ESP or TLS at each hop. This keying material is then used to generate the AES-CBC keys used to protect both RTP and RTCP messages between the two MTAs.

MTA_0 generates two randomly generated values: End-End $Secret_0$ (46-bytes) and Pad_1 (46-bytes).

MTA_1 generates two randomly generated values: End-End $Secret_1$ (46-bytes) and Pad_0 (46-bytes).

MTA_0 uses End-End $Secret_1$ and Pad_1 to derive encryption and authentication keys to be applied to its outbound traffic and used by MTA_1 to decrypt and authenticate it.

MTA_1 uses End-End $Secret_0$ and Pad_0 to derive encryption and authentication keys to be applied to its outbound traffic, and used by MTA_0 to decrypt and authenticate it. As a result, both MTA_0 and MTA_1 contribute randomly generated bytes to all of the keying material for both RTP and RTCP traffic.

The distribution of the end-to-end keying material is specific to the call signaling from [2] and is described in the following subsections.

7.6.2.3.1 Key Management over NCS

The diagram below shows the actual NCS messages that are used to carry out the distribution of end-to-end keys. Each NCS message that is involved in the end-to-end key management is labeled with a number of the corresponding key management interface.

The name of each NCS message is in bold. Below the NCS message name is the information needed in the NCS message, in order to perform end-to-end key distribution. Messages between the CMSs are labeled as SIP+ messages.

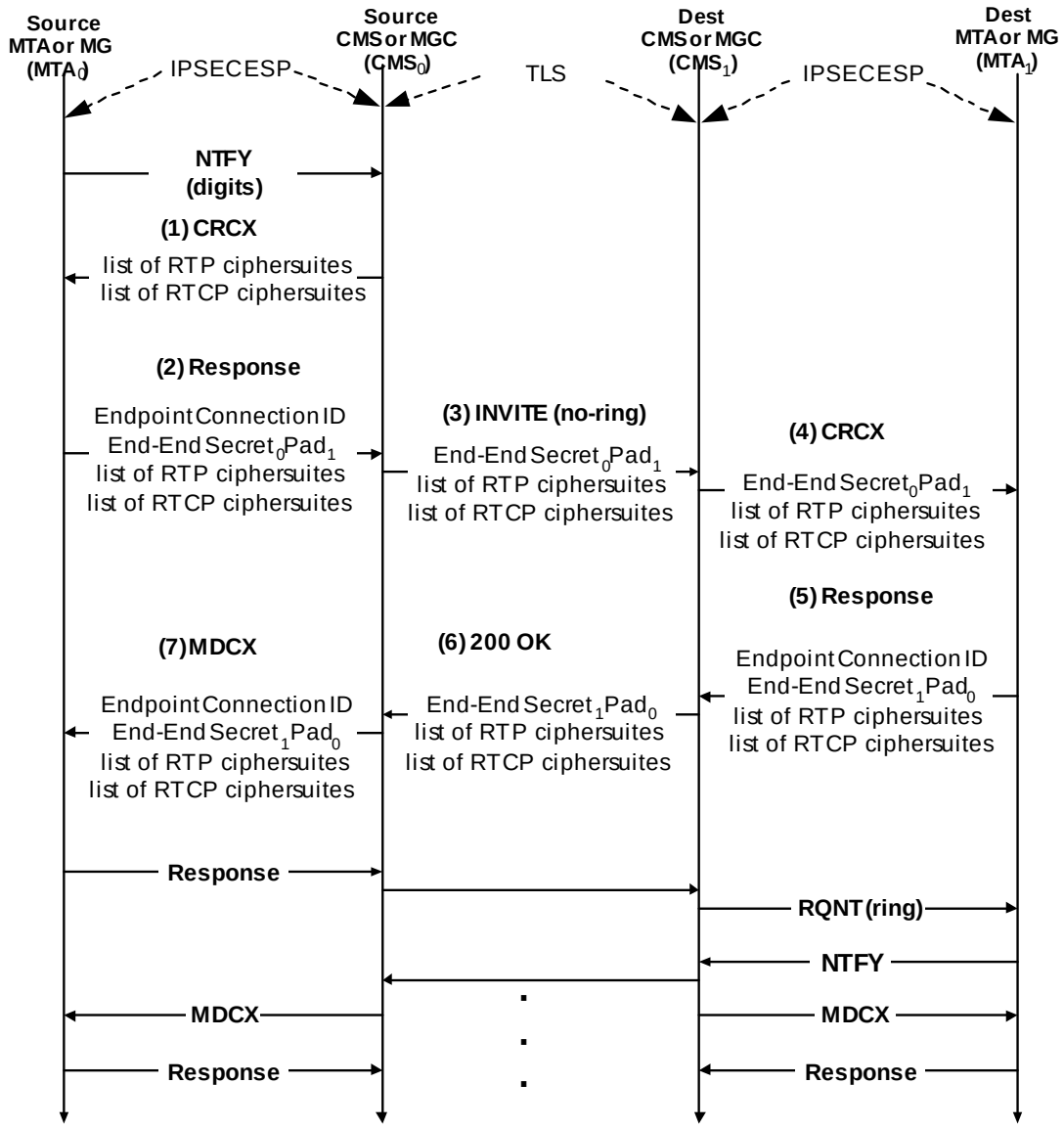


Figure 20. End-End Secret Distribution over NCS

This figure shows that before the start of this scenario, both the source and destination MTAs had already established an IPsec ESP session with their local CMS. It is also assumed that CMS-CMS signaling is secure.

This allows the End-End Secrets to be distributed securely, with privacy, integrity and anti-replay mechanisms already in place. The CMSs have access to this keying material but are trusted by the MTAs.

7.6.2.3.1.1 NULL Ciphersuite Combinations and Ordering

RTP_ENCR_NULL MUST only be used in conjunction with AUTH_NULL. RTP packets, with authentication but no encryption, are not allowed.

RTCP_AUTH_NULL MUST only be used in conjunction with RTCP_ENCR_NULL. RTCP messages with encryption and without authentication are not allowed.

Both RTP and RTCP security must be enabled or disabled together. The following five combinations MUST NOT be generated.

- RTP NULL encryption and RTP non-NULL authentication
- RTCP non-NULL encryption & RTCP NULL authentication
- RTP non-NULL encryption and RTCP NULL authentication
- RTP NULL encryption and RTCP non-NULL authentication
- RTP NULL encryption and RTCP non-NULL encryption.

If the MTA receives LocalConnectionOptions parameter that meet the above combinations, the MTA MUST return the error code 524 (Internal inconsistency in LocalConnectionOptions). Otherwise, if the MTA receives RemoteConnectionDescriptor parameter that meet the above combinations, then the MTA MUST return the error code 505 (Unsupported RemoteConnectionDescriptor).

For both RTP and RTCP ciphersuite lists exchanged during ciphersuite negotiation, the combination of NULL encryption and NULL authentication algorithms MUST always be included last. For example, the list of RTP ciphersuites "60/50;62/51;64/51" is not allowed, while the list of RTP ciphersuites "62/51;64/51;60/50", or "60/50" is allowed. If the list of ciphersuites in LocalConnectionOptions includes the NULL authentication and NULL encryption combination (60/50 for RTP, and 80/70 for RTCP), but this combination is not last, the MTA MUST return error code 524 (Internal inconsistency in LocalConnectionOptions). Otherwise, if this combination is not last in a RemoteConnectionDescriptor, error code 505 (Unsupported RemoteConnectionDescriptor) MUST be returned.

7.6.2.3.1.2 Ciphersuite Negotiation For MTAs

This specification only defines security for RTP/RTCP media streams, therefore ciphersuite negotiation applies only to RTP/RTCP media streams. Use of security for any other type of media streams is not specified.

An MTA MUST perform RTP and RTCP ciphersuite negotiation when processing any of the following:

- a CreateConnection command
- a ModifyConnection command with a RemoteConnectionDescriptor parameter
- a ModifyConnection command where the LocalConnectionOptions parameter includes ciphersuite fields.

An MTA MUST NOT perform ciphersuite negotiation in any other case. The steps involved in ciphersuite negotiation are the following:

1. An approved list of ciphersuites is formed by taking the intersection of the internal list of ciphersuites and ciphersuites allowed by the LocalConnectionOptions parameter, subject to the constraints specified in Section 7.6.2.3.1.1. The internal list of ciphersuites contains the ciphersuites that the MTA supports and which this specification requires. If the LocalConnectionOptions parameter was not included, or if the ciphersuite fields were not provided in the LocalConnectionOptions parameter, the approved list of ciphersuites contains the previously agreed upon approved list, or if no such list exists, the internal list of ciphersuites.
2. If the approved list of ciphersuites is empty, an error response MUST be generated, error code 532 (Unsupported value(s) in LocalConnectionOptions).
3. Otherwise, a negotiated list of ciphersuites is formed by taking the intersection of the approved list of ciphersuites and ciphersuites allowed by the RemoteConnectionDescriptor parameter (if present), subject to the constraints specified in Section 7.6.2.3.1.1. If a RemoteConnectionDescriptor was not provided, the negotiated list of ciphersuites thus contains the approved list of ciphersuites. If a RemoteConnectionDescriptor parameter is provided without fields containing the RTP and RTCP ciphersuite lists, then the RTP AUTH_NULL/RTP_ENCR_NULL and RTCP_AUTH_NULL/RTCP_ENCR_NULL ciphersuites

are assumed for the remote endpoints, and the regular ciphersuite negotiation process continues (i.e., the negotiated list of ciphersuites is formed by taking the intersection of the approved list of ciphersuites and the RTP AUTH_NULL/RTP_ENCR_NULL and RTCP_AUTH_NULL/RTCP_ENCR_NULL ciphersuites).

4. If the negotiated list of ciphersuites is empty, a ciphersuite negotiation failure has occurred and an error response MUST be generated. If a RemoteConnectionDescriptor parameter was provided, two different error codes can be returned:

    a. If the endpoint does not support any of the ciphersuites allowed by the RemoteConnectionDescriptor, error code 505 (Unsupported RemotedConnectionDescriptor) MUST be used.

    b. If the endpoint does support at least one of the ciphersuites, but the negotiated list of ciphersuites ended up being empty, error code 506 (Unable to satisfy both LocalConnectionOptions and RemoteConnectionDescriptor) MUST be used.

5. Otherwise, ciphersuite negotiation has succeeded, and the negotiated list of ciphersuites is returned in the LocalConnectionDescriptor parameter. Note that both LocalConnectionOptions and the RemoteConnectionDescriptor parameters can contain a list of ciphersuites that MUST be ordered by preference provided by the CMS in the RemoteConnectionDescriptor parameter. When both are supplied, the MTA SHOULD adhere to the preferences provided by the CMS in the RemoteConnectionDescriptor parameter, and otherwise, the MTA SHOULD adhere to the preferences provided in the LocalConnectionOptions parameter. If the MTA receives a RemoteConnectionDescriptor parameter with AUTH_NULL/RTP_ENCR_NULL for RTP or RTCP_AUTH_NULL/RTCP_ENCR_NULL for RTCP that is not last in the list, it MUST return the error code 505 (Unsupported RemoteConnectionDescriptor).

The following requirements apply during ciphersuite negotiation:

- A CMS MUST be capable of sending the allowable lists of ciphersuites for RTP and/or RTCP in the LocalConnectionOptions parameter of a CreateConnection command (CRCX) or a ModifyConnection command (MDCX) in the order of preference specified by the operator subject to the constraints specified in Section 7.6.2.3.1.1
- Whenever possible, a MTA SHOULD select the first supported ciphersuite for RTP and the first supported ciphersuite for RTCP in the RemoteConnectionDescriptor parameter. This allows the MTA to immediately start sending RTP and RTCP packets to the other MTA. An MTA MAY instead select alternate ciphersuites specified by the other MTA.
- When returning a LocalConnectionDescriptor and the negotiated list of RTP and RTCP ciphersuites is NULL, an MTA MUST NOT include an End-End Secret or Pad.
- When returning a LocalConnectionDescriptor and the negotiated list of RTP and RTCP ciphersuites contains at least one non-NULL selection each, an MTA MUST include an End-End Secret (for incoming RTP and RTCP packets) and MAY include a Pad value (for outgoing RTP and RTCP packets). The following rules apply:

    1. The MTA MUST generate a new End-End Secret when responding to a CreateConnection command.

    2. The MTA MUST generate a new End-End Secret when responding to a ModifyConnection command if the remote connection address (e.g., IP Address) or the remote transport address (e.g., port) are not identical to what was previously assigned.

    3. The MTA MUST use the existing End-End Secret when responding to a ModifyConnection command where there was no previous RemoteConnectionDescriptor provided.

    4. The MTA MUST generate a new Pad when responding to a CreateConnection command without a RemoteConnectionDescriptor.

5. The MTA MUST generate a new Pad when generating a new End-End Secret in response to a ModifyConnection command without a RemoteConnectionDescriptor.
6. If not otherwise required, the MTA MAY generate a new Pad when generating a new End-End Secret.
7. The MTA MUST NOT generate a new Pad when not generating a new End-End Secret.

- If, in response to a CreateConnection command, the list of ciphersuites selected for RTP contains at least one non-NULL encryption or authentication algorithm, before sending the response message, an MTA MUST:
  1. Establish inbound RTP security based on the preferred (first) RTP ciphersuite, its End-End Secret (which it generated), and a Pad value (if included in the RemoteConnectionDescriptor), as described in Section 7.6.2.3.3.1 of this specification.
  2. If a RemoteConnectionDescriptor was included and it contains media security attributes, establish outbound RTP security based on the selected RTP ciphersuite, End-End Secret (generated by the other MTA), and a Pad value (which it may have generated) as described in Section 7.6.2.3.3.1 of this specification.
  3. If connection mode allows, be ready to receive RTP packets, which may arrive any time after the Response message is sent.
- If, in response to a CreateConnection command, the list of ciphersuites for RTCP contains at least one non-NULL encryption algorithm, before sending the response message, an MTA MUST:
  1. Establish inbound RTCP security based on the preferred (first) RTCP ciphersuite, its End-End Secret (which it generated), and a Pad value (if included in the RemoteConnectionDescriptor), as described in Section 7.6.2.3.3.1 of this specification.
  2. If a RemoteConnectionDescriptor was included and it contained media security attributes, establish outbound RTCP security based on the selected RTCP ciphersuite, End-End Secret (generated by the far-end MTA), and a Pad value (which it may have generated) as described in Section 7.6.2.3.3.1 of this specification.
  3. Be ready to receive RTCP packets, which may arrive any time after the Response message is sent.
- If, in response to a ModifyConnection command that includes a RemoteConnectionDescriptor, and negotiated lists of ciphersuites for RTP and RTCP contain at least one non-NULL encryption or authentication algorithm each, before sending the response message, an MTA MUST:
  1. If a Pad was included in the RemoteConnectionDescriptor and it is different than a Pad that may have previously been received, remove any existing inbound RTP keys and generate new ones, based on the keys that are generated from both the End-End Secret (generated locally) and the Pad (generated by the other MTA). The MTA MUST re-initialize the RTP timestamp if new keys are generated The ciphersuites used for these inbound keys are taken from the RemoteConnectionDescriptor parameter just received from the CMS.
  2. If a Pad was included in the RemoteConnectionDescriptor and it is different than a Pad that may have previously been received, remove any existing inbound RTCP keys and generate new ones, based on the keys that are generated from both the End-End Secret (generated locally) and the Pad (generated by the other MTA). The MTA MUST re-initialize RTCP sequence numbers if new keys are generated. The ciphersuites used for these inbound keys are taken from the RemoteConnectionDescriptor parameter just received from CMS.
  3. If the RemoteConnectionDescriptor parameter was received without a Pad, check if the first RTP ciphersuite field in the RemoteConnectionDescriptor parameter differs from the one that the MTA originally selected. Also, check to see if a Pad had been previously received. If the ciphersuites differ, or if a Pad had been previously received, perform the following steps:
     a. Remove any existing inbound RTP key.

b. If the new RTP ciphersuite is non-NULL, generate new inbound RTP keys and RTP timestamp from the same End-End Secret (generated locally) as the last time, as specified in Section 7.6.2.3.3.1.

4. If the RemoteConnectionDescriptor parameter was received without a Pad, check if the first RTCP ciphersuite field in the RemoteConnectionDescriptor parameter differs from the one that the MTA originally selected. Also, check to see if a Pad had been previously received. If the ciphersuites differ, or if a Pad had been previously received, perform the following steps:

   a. Remove any existing inbound RTCP key.

   b. If the new RTCP ciphersuite is non-NULL, generate new inbound RTCP keys from the same End-End Secret (generated locally) as the last time, as specified in Section 7.6.2.3.3.1, and reset the RTCP sequence number to 0.

5. If the End-End Secret included in the RemoteConnectionDescriptor has changed or the negotiated RTP ciphersuite has changed, perform the following steps:

   a. Remove any existing outbound RTP keys.

   b. If the new list of RTP ciphersuites is non-NULL, generate new outbound RTP keys, based on the End-End Secret (generated by the other MTA) and the Pad (generated locally), and generate a new RTP timestamp.

6. If the End-End Secret included in the RemoteConnectionDescriptor has changed or the negotiated RTCP ciphersuite has changed, perform the following steps:

   a. Remove any existing outbound RTCP keys

   b. If the new list of RTCP ciphersuites is non-NULL, generate new outbound RTCP keys, based on the End-End Secret (generated by the other MTA) and the Pad (generated locally), and reset the RTCP sequence number to 0.

7. Be ready to send RTCP messages to and receive RTCP messages from the remote MTA. If connection mode allows, be ready to send and receive RTP messages with the remote MTA. If the list of ciphersuites for RTP was sent within a ModifyConnection command, the CMS MAY send an inactive directive to the MTA in the same command. The MTA should be returned to active status only when the new ciphersuite negotiation is complete.

- If, in response to a ModifyConnection command that does not include a RemoteConnectionDescriptor, and negotiated lists of ciphersuites for RTP and RTCP contain at least one non-NULL encryption or authentication algorithm each, before sending the response message, an MTA MUST:

  1. If the first RTP ciphersuites field in the negotiated list differs from the one that the MTA previously selected, then perform the following steps:

     a. Remove any existing inbound RTP keys.

     b. Generate new inbound RTP keys from the previous End-End Secret (locally generated) and Pad (generated by the other MTA), and generate a new RTP timestamp.

  2. If the first RTCP ciphersuites field in the negotiated list differs from the one that the MTA previously selected, then perform the following steps:

     a. Remove any existing inbound RTCP keys.

     b. Generate new inbound RTCP keys from the previous End-End Secret (locally generated) and Pad (generated by the other MTA), and reset the RTCP sequence number to 0.

  3. Be ready to send RTCP messages to and receive RTCP messages from the remote MTA. If connection mode allows, be ready to send and receive RTP messages with the remote MTA. If the list of ciphersuites for RTP was sent within a ModifyConnection command, the CMS

MAY send an inactive directive to the MTA in the same command. The MTA should be returned to active status only when the new ciphersuite negotiation is complete.

- If an MTA receives a ModifyConnection command, and the resulting intersection of ciphersuites results in NULL encryption and authentication algorithms for RTP and RTCP, then the MTA MUST remove any existing RTP and RTCP keys and do not perform security on the RTP and RTCP packets.
- If an MTA returns a LocalConnectionDescriptor parameter, it MUST return the latest negotiated list of ciphersuites.

The following message flow is informative. Each of the numbered flows in Figure 20 is described below:

(1) CMS_0 -> MTA_0

CMS_0 may send the allowable lists of ciphersuites for the new communication to MTA_0 in the CreateConnection (CRCX) command, inside the LocalConnectionOptions parameter, if the CMS has been configured to do so. The ciphersuites are provided in the order of preference specified by the operator subject to the constraints specified in Section 7.6.2.3.1.1. There can be two lists of ciphersuites, one list for RTP security and one for RTCP security. Each of these two lists may be included to specify the list of allowable ciphersuites, however ciphersuite negotiation will take place for both RTP and RTCP irrespective of whether the lists are included or not.

If RTP and/or RTCP ciphersuites are included but do not adhere to the rules provided in Section 7.6.2.3.1.1, the MTA returns an error, e.g. 524 (Internal inconsistency in LocalConnectionOptions).

(2) MTA_0 -> CMS_0

MTA_0 performs ciphersuite negotiation according to the ciphersuite negotiation procedure described above, and returns a non-empty list of RTP ciphersuites in the response message. This list contains the list of MTA_0's list of allowed ciphersuites in the order of preference specified by CMS_0 if the LocalConnectionOptions ciphersuites parameter(s) is included in step (1), as specified above. If RTP or RTCP ciphersuite negotiation fails, MTA_0 returns an error code as specified above.

If the lists of negotiated ciphersuites for RTP and RTCP contain at least one non-NULL combination each, MTA_0 generates the End-End $Secret_0$ and Pad_1 value and returns them along with the ciphersuites in the LocalConnectionDescriptor parameter. For further details on the NCS message syntax, refer to [2]. Note that the NULL authentication and NULL encryption combinations will be at the end of each ciphersuite list.

The response message also includes the ConnectionId and the EndpointId for MTA_0 as described in [2]. The pair (ConnectionId, EndpointId) uniquely identifies this connection, where the EndpointId is an NCS identifier for MTA_0.

If the list of ciphersuites for RTP contains at least one non-NULL encryption or authentication algorithm, before sending the response message, MTA_0 must:

1. Establish inbound RTP security based on its preferred (first) RTP ciphersuite and End-End $Secret_0$, as described in Section 7.6.2.3.3.1 of this specification.
2. If connection mode allows, be ready to receive RTP packets, which may arrive any time after this message is sent by the MTA_0. If the list of ciphersuites for RTP was sent within a ModifyConnection command, the CMS may send an inactive directive to the MTA in the same command. The MTA should be returned to active status only when the new ciphersuite negotiation is complete.

If the list of ciphersuites for RTCP contains at least one non-NULL encryption algorithm, before sending the response message, MTA_0 must:

1. Establish inbound RTCP security based on its preferred (first) RTCP ciphersuite and End-End $Secret_0$, as described in Section 7.6.2.3.3.1 of this specification.

2. Be ready to receive RTCP packets, which may arrive any time after this message is sent by MTA_0.

If MTA_1 decides to use an alternate ciphersuite listed by MTA_0, MTA_0 will later have to update its RTP and RTCP keys. If MTA_1 decides to send MTA_0 packets before ciphersuite negotiation had completed, processing on those packets at MTA_0 will fail (since it assumed a different ciphersuite). If media stream security is disabled (AUTH_NULL/RTP_ENCR_NULL ciphersuite list for RTP and RTCP_AUTH_NULL/RTCP_ENCR_NULL for RTCP), MTA_0 will later have to discard its keys and send and receive RTP and RTCP packets without any security.

(3) CMS_0 -> CMS_1

CMS_0 must send End-End $Secret_0$ (if included), Pad_1 (if included) and the list of RTP and RTCP ciphersuites to CMS_1 (local to MTA_1) as selected by MTA_0. CMS_1 will later forward this information to MTA_1. Note that End-End $Secret_0$ and Pad_1 will not be included if the RTP and RTCP ciphersuites lists both contain only the NULL authentication and NULL encryption combination.

(4) CMS_1 -> MTA_1

CMS_1 sends a CreateConnection to MTA_1. CMS_1 may provide lists of approved RTP and RTCP ciphersuites, if the CMS has been configured to do so. The ciphersuites are provided in the order of preference specified by the operator subject to the constraints specified in Section 7.6.2.3.1.1. The RemoteConnectionDescriptor must be included in this CRCX command. It must contain End-End $Secret_0$ (if sent in step (3))and Pad_1 (if sent in step (3)) received from MTA_0 (via CMS_0). It must also contain the ciphersuites preferred by MTA_0.

(5) MTA_1 -> CMS_1

MTA_1 has received a CRCX message that contains both LocalConnectionOptions and RemoteConnectionDescriptor parameters and must follow the ciphersuite negotiation procedure described above to negotiate RTP and RTCP ciphersuites. This list will consist of MTA_1's allowed ciphersuites in the order of preference specified by CMS_1 if the LocalConnectionOptions ciphersuites parameter is included in step (4). If RTP and RTCP ciphersuite negotiation succeeds and there is at least one RTP ciphersuite and at least one RTCP ciphersuite, then MTA_1 returns the negotiated list of ciphersuites in the subsequent response message, in the LocalConnectionDescriptor parameter, in the form of SDP attributes. Note that if media stream security is being disabled, the NULL authentication and NULL encryption combination will be the only entry in both the RTP and RTCP ciphersuites lists. If RTP or RTCP ciphersuite negotiation fails, MTA_1 must return an error code as specified above.

In the event that MTA_1 receives SDP in the RemoteConnectionDescriptor parameter without ciphersuites media attributes, MTA_1 assumes that the lists of RTP and RTCP ciphersuites supported by the remote endpoint is RTP AUTH_NULL/RTP_ENCR_NULL and RTCP_AUTH_NULL/RTCP_ENCR_NULL.

If the RTP and RTCP ciphersuites provided do not adhere to the rules provided in Section 7.6.2.3.1.1, the MTA returns an error, e.g. 524 (Internal inconsistency in LocalConnectionOptions).

Whenever possible, MTA_1 should select the first supported ciphersuite for RTP and the first supported ciphersuite for RTCP in the RemoteConnectionDescriptor parameter. This allows MTA_1 to immediately start sending RTP and RTCP packets to MTA_0. MTA_1 may instead select alternate ciphersuites specified by MTA_0. MTA_1 returns a response message, which includes lists of the selected ciphersuites inside the LocalConnectionDescriptor parameter, in the form of SDP attributes. The first ciphersuite in each list (one for RTP and one for RTCP) must be the one that was selected by MTA_1. Additional ciphersuites in each list are alternatives in a prioritized order. If at any time, MTA_0 wants to switch to one of the alternatives that were selected by MTA_1, it would have to go through a new key negotiation. The response message must also include the ConnectionId (generated by MTA_1) as specified in [2]. Thus, both End-End $Secret_0$ and End-End $Secret_1$ are now associated with a pair (EndpointId, ConnectionId).

If the lists of ciphersuites for RTP and RTCP contain at least one non-NULL selection each, then MTA_1 must generate the End-End $Secret_1$ for the incoming RTP and RTCP packets, MTA_1 must and return it along with the ciphersuite lists in the LocalConnectionDescriptor parameter. If the lists of ciphersuites for RTP and RTCP contain at least one non-NULL selection each, MTA_1 should also generate Pad_0 and return it in the same LocalConnectionDescriptor parameter.

Although the option of not generating Pad_0 is provided in order to better support early media flows from MTA_1, it results in MTA_1 using a send key that is completely dependent on a random value generated by MTA_0. In other words, privacy of the media stream generated by MTA_1 in this case depends on the strength of MTA_0's random number generator.

If the list of ciphersuites for RTP contains at least one non-NULL encryption or authentication algorithm, before sending the response message, MTA_1 must:

1. Establish inbound RTP security based on its selected RTP ciphersuite, End-End $Secret_1$ and Pad_1, as described in Section 7.6.2.3.3.1 of this specification.
2. Establish outbound RTP security based on its selected RTP ciphersuite and End-End $Secret_0$, as described in Section 7.6.2.3.3.1 of this specification. If Pad_0 was generated by MTA_1, the outbound RTP security will also be based on Pad_0.
3. If connection mode allows, be ready to receive RTP packets, which may arrive from MTA_0 any time after this message is sent.

If the list of ciphersuites for RTCP contains at least one non-NULL encryption or authentication algorithm, before sending the response message, MTA_1 must:

1. Establish inbound RTCP security based on its selected RTCP ciphersuite, End-End $Secret_1$ and Pad_1 as described in Section 7.6.2.3.3.1 of this specification.
2. Establish outbound RTCP security based on its selected RTCP ciphersuite and End-End $Secret_0$, as described in Section 7.6.2.3.3.1 of this specification. If Pad_0 was generated by MTA_1, the outbound RTCP security will also be based on Pad_0.
3. Be ready to receive RTCP messages, which may arrive from MTA_0 any time after this message is sent.

Any time after sending this response message to the CMS_1, MTA_1 may begin sending RTP and RTCP packets to MTA_0. However, in the case that MTA_1 generated Pad_0 or selected a different ciphersuite from the one preferred by MTA_0, MTA_0 will not be able to decrypt packets from MTA_1, until MTA_0 has received MTA_1's SDP.

(6) CMS_1 -> CMS_0

CMS_1 must forward the End-End $Secret_1$, (if included) Pad_0 (if included) and the selected ciphersuites sent from MTA_1 to CMS_0. Note that End-End $Secret_0$ and Pad_1 will not be included if the RTP and RTCP ciphersuites lists both contain only the NULL authentication and NULL encryption algorithm combination.

(7) CMS_0 -> MTA_0

CMS_0 may send to MTA_0 in the ModifyConnection command, inside the LocalConnectionOptions parameter, the lists of allowed RTP and RTCP ciphersuites. These ciphersuites should be what CMS_0 policy allows. (The reason that CMS_0 is not required to send the lists of ciphersuites is because it might have already sent them to MTA_0 in a CreateConnection command. CMS_0 would send the ciphersuites again for consistency

In the event that MTA_0 receives SDP in the RemoteConnectionDescriptor parameter without fields containing ciphersuites media attributes, MTA_0 assumes that the RTP and RTCP ciphersuite lists supported by the remote endpoint are AUTH_NULL/RTP_ENCR_NULL for RTP and RTCP_AUTH_NULL/RTCP_ENCR_NULL for RTCP.

In the event that CMS_0 received SDP from MTA_1, the RemoteConnectionDescriptor parameter must be included in this ModifyConnection command. If present, it must contain the RTP and

RTCP ciphersuites (and alternatives) selected by MTA_1. If ciphersuites are included in the LocalConnectionOptions parameter or a RemoteConnectionDescriptor parameter is included with the ModifyConnection command, MTA_0 must perform ciphersuite negotiation as described above.

If the RemoteConnectionDescriptor is not sent in this MDCX command, MTA_0 will still be able to receive RTP and RTCP messages but will be unable to send anything to MTA_1.

After receiving this message, MTA_0 must:

1. If Pad_0 was received, remove its inbound RTP keys and replace them with new ones, based on the keys that are generated from both End-End $Secret_0$ and Pad_0. Re-initialize the RTP timestamp for the new keys. The ciphersuites used for these inbound keys are taken from the RemoteConnectionDescriptor just received from CMS_0.
2. If Pad_0 was received, remove its inbound RTCP keys and replace them with new ones, based on the keys that are generated from both End-End $Secret_0$ and Pad_0. Re-initialize RTCP sequence numbers for the new keys. The ciphersuites used for these inbound keys are taken from the RemoteConnectionDescriptor just received from CMS_0.
3. If the RemoteConnectionDescriptor was received without Pad_0, check if the first RTP ciphersuite in the RemoteConnectionDescriptor differs from the one that MTA_0 selected in step (2). If they differ, perform the following steps:
    a. Remove the inbound RTP key.
    b. If the new RTP ciphersuite is non NULL, generate new inbound RTP keys and RTP timestamp from the same End-End $Secret_0$ as the last time, as specified in Section 7.6.2.3.3.1.
4. If the RemoteConnectionDescriptor parameter was received without Pad_0, check if the first RTCP ciphersuite field in the RemoteConnectionDescriptor parameter differs from the one that MTA_0 selected in step (2). If they differ, perform the following steps:
    a. Remove the inbound RTCP key.
    b. If the new RTCP ciphersuite is non NULL, generate a new key based on the key generated from the same End-End $Secret_0$ as the last time, but for the new authentication and/or encryption algorithms.
5. If the RemoteConnectionDescriptor parameter was received, establish outbound RTP keys, based on End-End $Secret_1$ and Pad_1.
6. If the RemoteConnectionDescriptor parameter was received, establish outbound RTCP keys, based on End-End $Secret_1$ and Pad_1.
7. Be ready to send and receive RTCP messages with MTA_1. If connection mode allows, be ready to send and receive RTP messages with MTA_1.

For full syntax of the NCS messages, please refer to the NCS signaling specification [2].

7.6.2.3.2 Ciphersuite Format

Each ciphersuite for both RTP security and RTCP security MUST be represented as follows:

Authentication Algorithm (1 byte) – represented by 2 ASCII hex characters (using characters 0-9, A-F).	**Encryption Transform ID** (1 byte) – represented by 2 ASCII hex characters (using characters 0-9, A-F).

For the list of available transforms and their values, refer to Section 6.6 for RTP security and 6.7 for RTCP security. For the exact syntax of how the Authentication Algorithm and the Encryption Transform ID are included in the signaling messages, refer to [2] for NCS.

7.6.2.3.3 Derivation of End-to-End Keys

7.6.2.3.3.1 Initial Key Derivation

The End-End Secrets MUST be 46 bytes long. The Pad parameters MUST be 46 bytes long.

Keys are independently derived by each MTA from either just the End-End Secret or from the End-End Secret and Pad concatenated together. The Pad may or may not be available – see the call flow details specified in Section 7.6.2.3.1.

The keys derived from one End-End Secret (and possibly a Pad) MUST be used to secure RTP and RTCP messages directed to only one of the MTAs. There is a separate End-End Secret and a separate Pad value for each direction, negotiated through NCS signaling. The keys MUST be derived as follows, in the specified order:

1. RTP (media stream security). Derive a set of the following keys with the derivation function F(S, "End-End RTP Security Association"). Here, S is concatenation of the following binary values, each in MSB-first order:

    a. End-End Secret

    b. Pad (optional, if it was negotiated through signaling)

    The string "End-End RTP Security Association" is taken without quotes and without a terminating null character. Function F (specified in Section 9.6) is used to recursively generate enough random bytes to produce all of the keys and other parameters that are specified below, in the listed order:

    a. RTP privacy key.

    b. RTP Initial Timestamp (integer value, 4 octets, Big Endian byte order

    c. RTP Initialization Key (required when using a block cipher to encrypt the RTP payload). The length MUST be the same as the selected cipher's block size. This value is used to derive the IV according to 7.6.2.1.2.2. The resulting IV is used for the block cipher in CBC mode (if applicable) and for the random pad used to calculate the MMH-MAC.

    d. RTP packet MAC key (if MAC option is selected). The requirements for the MMH MAC key can be found in Section 7.6.2.1.2.1.1.

2. RTCP security. Derive a set of the following keys in the specified order with the derivation function F(S, "End-End RTP Control Protocol Security Association"). Here, S is concatenation of the following binary values:

    a. End-End Secret

    b. Pad (optional, if it was negotiated through signaling)

    Function F (specified in Section 9.6) is used to recursively generate enough random bytes to produce all of the keys that are specified below, in the listed order:

    a. RTCP authentication key.

    b. RTCP encryption key.

7.6.2.4 RTP-RTCP Summary Security Profile Matrix

Table 27. Security Profile Matrix – RTP & RTCP

	RTP (MTA – MTA, MTA – MG)	RTCP (MTA – MTA, MTA – MG, MG – MG)
authentication	optional (indirect)[24]	optional (indirect)
access control	optional	optional
integrity	optional	optional
confidentiality	optional	optional
non-repudiation	no	no
security mechanisms	Application Layer Security via RTP PacketCable Security Profile End-to-End Secret distributed over secured MTA-CMS links. Final keys derived from this secret. AES-128 in CBC mode encryption algorithm Optional 2-byte or 4-byte MAC based on MMH algorithm RTP encryption and authentication can be optionally turned off with the selection of NULL encryption and NULL authentication algorithms. RTP security and RTCP security are disabled together. PacketCable requires support for ciphersuite negotiation.	RTCP messages are secured by RTCP application layer security mechanisms specified in the profile. RTCP ciphersuites are negotiated separately from the RTP ciphersuites and include both encryption and message authentication algorithms. RTCP encryption can be optionally turned off with the selection of a null encryption algorithm. Both RTCP encryption and authentication can be optionally turned off with the selection of NULL encryption and NULL authentication algorithms. RTCP security and RTP security are disabled together. Keys are derived from the end-end secret using the same mechanism as used for RTP encryption

7.7 Audio Server Services

7.7.1 Reference Architecture

The following diagram shows the network components and the various interfaces to be discussed in this Section, see [27].

[4] MTAs do not authenticate directly. Authentication refers to the authentication of identity.

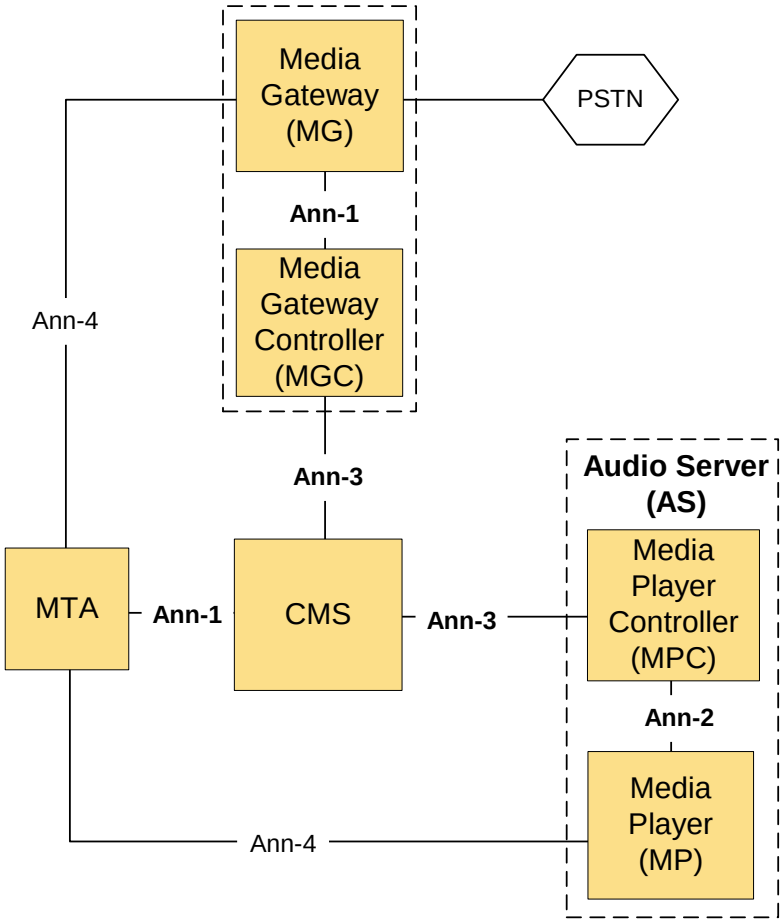


Figure 21. Audio Server Components and Interfaces

Figure 21 shows a network-based Media Player (MP). It has an optional Audio Server Protocol (ASP) interface (Ann-2) to the Media Player Controller (MPC), in the case that MPC and MP are not integrated into a single physical entity. Security on this interface is specified in this section.

There is also an NCS signaling interface (Ann-1) between the MTA and CMS and between the Media Gateway Controller (MGC) and the Media Gateway (MG). Refer to Section 7.4.1 for NCS signaling security. There is also a signaling interface (Ann-3) between the CMS and the MPC and the CMS and the MGC. This interface is proprietary for PacketCable, and thus the corresponding security interface is not specified (although this section lists recommended security services for Ann-3).

Finally, there is a media stream (RTP and RTCP) interface (Ann-4) between the MTA and the MP. This is a standard media stream interface, for which security is defined in Section 6.6 of this document.

The Audio Server Architecture also allows local playout of announcements at the MTA. In those cases, an announcement is initiated with NCS signaling between the MTA and the CMS (interface Ann-1). No other interfaces are needed for MTA-based announcement services.

7.7.2 Security Services

7.7.2.1 MTA-CMS NCS Signaling (Ann-1)

Refer to the security services in the NCS signaling Section 7.4.1.2 of this document.

7.7.2.2 MPC-MP Signaling (Ann-2)

Authentication: all signaling messages must be authenticated, in order to prevent a third party masquerading as either an authorized MPC or MP. A rogue MPC could configure the MP to play obscene or inappropriate messages. A rogue MP could likewise play obscene or inappropriate messages that the MPC did not intend it to play. If MP is unable to authenticate to the MPC, the MPC should not pass it the key for media packets, preventing unauthorized announcement playout.

Confidentiality: if a snooper is able to monitor ASP signaling messages on this interface, he or she might determine which services are used by a particular subscriber or which destinations a subscriber is communicating to. This information could then be sold for marketing purposes or simply used to spy on other subscribers. Thus, confidentiality is required on this interface.

Message integrity: must be assured in order to prevent tampering with signaling messages. This could lead to playout of obscene or inappropriate messages – see authentication above.

Access control: an MPC should keep a list of valid Media Players and which announcements each supports. Along with authentication, this insures that wrong announcements are not played out.

7.7.2.3 MTA-MP (Ann-4)

Security services on this media packet interface are listed in Section 7.6.1.

7.7.3 Cryptographic Mechanisms

7.7.3.1 MTA-CMS NCS Signaling (Ann-1)

Refer to the cryptographic mechanisms in the NCS signaling Section 7.4.1.3 of this document.

7.7.3.2 MPC-MP Signaling (Ann-2)

IPsec ESP MUST be used to both authenticate and encrypt the messages from MPC to MP and vice versa. Refer to Section 6.1.2 for details of how IPsec ESP is used within PacketCable and for the list of available ciphersuites.

7.7.3.3 MTA-MP (Ann-4)

Cryptographic mechanisms on this media packet interface are specified in Section 7.6.2.

7.7.4 Key Management

7.7.4.1 MTA-CMS NCS Signaling (Ann-1)

Refer to the key management in the NCS signaling Section 7.4.1.

7.7.4.2 MPC-MP Signaling (Ann-2)

The MPC and the MP negotiate a shared secret (MPC-MP Secret) using IKE or Kerberos (implementations MUST support IKE with pre-shared keys; they MAY support IKE with X.509 certificates and they MAY support Kerberos using symmetric keys). For more information on the PacketCable use of IKE, refer to Section 6.2.2. For more information on the PacketCable use of Kerberos with symmetric keys, refer to Sections 6.4.3 and 6.5.

The key management protocol MUST be running asynchronous to the signaling messages and will guarantee that there is always a valid, non-expired MPC-MP Secret.

7.7.4.3 MTA-MP (Ann-4)

Key Management on the media packet interface is specified in Section 7.6.2.3. This case is very similar to the key management for the MTA-MG media interface. The flow of signaling messages and the syntax of

carrying keys and ciphersuites MUST be the same, except that here MG is replaced with the MP and MGC (which delivers the key to MG) is replaced with MPC (which delivers the key to MP).

7.7.5 MPC-MP Summary Security Profile Matrix

The CMS to MPC protocol is not defined in PacketCable and thus is outside the scope of this document. The corresponding column in the following matrix provides only the security requirements on that interface. Security specifications on that interface will be added in future revisions of this document.

Table 28. Security Profile Matrix – Audio Server Services

	Ann-1: NCS (MTA – CMS) & (MG – MGC)	**Ann-2: ASP (MPC-MP)**	**Ann-3: unspecified (CMS-MPC) & (CMS – MGC) Interface Security Requirement**	**Ann-4: RTP (MTA-MP)**	**Ann-4: RTCP (MTA-MP)**
authentication	yes	yes	yes	yes (indirect)	yes (indirect)
access control	yes	yes	yes	optional	optional
Integrity	yes	yes	yes	optional	yes
confidentiality	yes	yes	yes	yes	yes
non-repudiation	no	no	no	no	no
security mechanisms	IPsec ESP in transport mode, encryption and message integrity both enabled Kerberos with PKINIT key management for MTA – CMS interface IKE or Kerberos for MG – MGC interface	IPsec IKE or Kerberos		Application Layer Security via RTP Packet Cable Security Profile keys distributed over secured MTA-CMS and MP-MPC links AES-128 encryption algorithm Optional 2-byte or 4-byte MAC based on MMH algorithm.	RTCP messages are secured by RTCP application layer security mechanisms specified in the profile. Keys are derived from the end-end secret using the same mechanism as used for RTP encryption.

*Although (CMS – MPC) is a proprietary interface, the following are security requirements for the CMS-MPC interface.

7.8 Electronic Surveillance Interfaces

7.8.1 Reference Architecture

The PacketCable system for Electronic Surveillance (see [30]) consists of the following elements and interfaces:

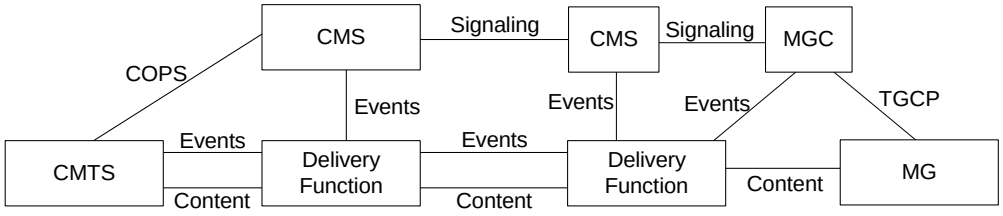


Figure 22. Electronic Surveillance Security Interfaces

The DF (Delivery Function) in this diagram is responsible for redirecting duplicated media packets to law enforcement, for the purpose of wiretapping.

The event interface between the CMS or the MGC and the DF provides descriptions of calls, which is necessary to perform wiretapping That information includes the media stream encryption key and the corresponding encryption algorithm. This event interface uses RADIUS and is similar to the CMS-RKS interface.

The COPS interface between the CMS and the CMTS is used to signal the CMTS to start/stop duplicating media packets to the DF for a particular call. This is the same COPS interface that is used for (DQoS) Gate Authorization messages. For the corresponding security services, refer to Sections 7.2.1.2.2, 7.2.1.3.2 and 7.2.1.4.1.

The TGCP signaling interface between the MGC and MG is used to signal the MG to start/stop duplicating media packets to the DF for a particular call. This is the same TGCP signaling interface that is used during call setup on the PSTN Gateway side. For the corresponding security services, refer to Sections 7.8.2.1, 7.8.3.1 and 7.8.4.1.

The event interface between the CMTS and DF is needed to tell the DF when the actual call begins and when it ends. In PacketCable, the start and end of the actual call is signaled with RADIUS event messages generated by the CMTS.

The interface between the CMTS and DF for call content is where the CMTS encapsulates copies of the RTP media packets – including the original IP header –inside UDP and forwards them to the DF. Since the original media packets are already encrypted (and optionally authenticated), no additional security is defined on this interface. Similarly, there is no additional security applied to the call content interface between the MG and DF: the MG simply encapsulates copies of the encrypted RTP packets inside UDP and forwards them to the DF.

The event interface between the two DFs is used to forward call information in the case where a wiretapped call is forwarded to another location that is wiretapped using a different DF. This interface utilizes the RADIUS protocol – the same as all other event message interfaces.

The interface between the two DFs for call content is used to forward media packets (including the original IP header) in the case where a wiretapped call is forwarded to another location that is wiretapped using a different DF. Since the original media packets are already encrypted (and optionally authenticated), no additional security is defined on this interface.

7.8.2 Security Services

7.8.2.1 Event Interfaces CMS-DF, MGC-DF, CMTS-DF and DF-DF

Authentication, Access Control and Message Integrity: required to prevent service theft and denial-of-service attacks. Want to insure that the DF (law enforcement) has the right parameters for wiretapping (prevent denial-of-service). Also, want to authenticate the DF, to make sure that the copy of the media stream is directed to the right place (protect privacy).

Confidentiality: required to protect subscriber information and communication patterns.

7.8.2.2 Call Content Interfaces CMTS-DF, MG-DF, MG-DF and DF-DF

Authentication and Access Control: already performed during the phase of key management for protection of event messages – see the above section. In order to protect privacy, a party that is not properly authorized should not receive the call content decryption key.

Message Integrity: optional for voice packets, since it is generally hard to make undetected changes to voice packets. No additional security is required here – an optional integrity check would be placed into the media packets by the source (MTA or MG).

Confidentiality: required to protect call content from unauthorized snooping. However, no additional security is required in this case – the packets had been previously encrypted by the source (MTA or MG).

7.8.3 Cryptographic Mechanisms

7.8.3.1 Interface between CMS and DF

This interface MUST be protected with IPsec ESP in transport mode, where each packet is both encrypted and authenticated – identical to the security for the CMS–RKS interface specified in Section 7.3.3.1.

As with the CMS-RKS interface, the MAC value normally used to authenticate RADIUS messages is not used (message integrity is provided with IPsec). The key for this RADIUS MAC MUST always be hardcoded to 16 ASCII 0s.

7.8.3.2 Interface between CMTS and DF for Event Messages

This interface MUST be protected with IPsec ESP in transport mode, where each packet is both encrypted and authenticated – identical to the security for the CMTS–RKS interface specified in Section 7.3.3.2.

As with the CMTS-RKS interface, the MAC value normally used to authenticate RADIUS messages is not used (message integrity is provided with IPsec). The key for this RADIUS MAC MUST always be hardcoded to 16 ASCII 0s.

7.8.3.3 Interface between DF and DF for Event Messages

This interface MUST be protected with IPsec ESP in transport mode, where each packet is both encrypted and authenticated – identical to the security for the CMS–RKS interface specified in Section 7.3.3.1.

As with the CMS-RKS interface, the MAC value normally used to authenticate RADIUS messages is not used (message integrity is provided with IPsec). The key for this RADIUS MAC MUST always be hardcoded to 16 ASCII 0s.

7.8.3.4 Interface between MGC and DF

This interface MUST be protected with IPsec ESP in transport mode, where each packet is both encrypted and authenticated – identical to the security for the MGC–RKS interface specified in Section 7.3.3.3.

As with the MGC-RKS interface, the MAC value normally used to authenticate RADIUS messages is not used (message integrity is provided by IPsec). The key for this RADIUS MAC MUST always be hardcoded to 16 ASCII 0s.

7.8.4 Key Management

7.8.4.1 Interface between CMS and DF

The CMS and the DF MUST negotiate a pair of IPsec Security Associations (inbound and outbound) using IKE or Kerberos (implementations MUST support IKE with pre-shared keys; they MAY support IKE with X.509 certificates and they MAY support Kerberos using symmetric keys). For more information on the PacketCable use of IKE, refer to Section 6.2.2. For more information on the PacketCable use of Kerberos with symmetric keys, refer to Sections 6.4.3 and 6.5.

The key management protocol will be running asynchronous to the event message generation, and will guarantee that there is always a valid, non-expired pair of Security Associations.

7.8.4.2 Interface between CMTS and DF

The CMTS and the DF MUST negotiate a pair of Security Associations (inbound and outbound) using IKE or Kerberos (implementations MUST support IKE with pre-shared keys; they MAY support IKE with X.509 certificates and they MAY support Kerberos using symmetric keys). For more information on the PacketCable use of IKE, refer to Section 6.2.2. For more information on the PacketCable use of Kerberos with symmetric keys, refer to Sections 6.4.3 and 6.5.

The key management protocol will be running asynchronous to the event message generation, and will guarantee that there is always a valid, non-expired pair of Security Associations.

7.8.4.3 Interface between DF and DF

The two DF hosts MUST negotiate a shared secret (DF-DF Secret) using IKE with certificates. The PacketCable profile for IKE with certificates is specified in Section 6.2.2. IKE will be running asynchronous to the event message generation. In the case where an event message needs to be sent to a DF with which there is not a valid SA, the IPsec layer MUST automatically signal IKE to proceed with the key management exchanges and build a pair of IPsec SAs (inbound and outbound).

Not all interfaces between the same pair of DFs will require IPsec. For example, the call content interface does not run over IPsec. In order for the IPsec SAs to be established only for the DF-DF event message interface, each DF MUST allocate a set of UDP ports on which it will both send and receive DF-DF event messages. IPsec policy database for each DF MUST specify either an enumeration or a range of local UDP ports for which IPsec is enabled and which will be used exclusively for DF-DF event messages. If there are multiple calls that are simultaneously wiretapped and forwarded between the same pair of DFs (on different UDP ports) – they MUST all be protected with a single pair of IPsec SAs (inbound-outbound). Whenever a DF attempts to send on one of those UDP ports, it will either use an existing IPsec SA for a particular destination DF, or it will trigger IKE to establish a pair of SAs (inbound-outbound) for the specific target DF. When the CMS tells a DF to forward event messages to another DF, it specifies the destination DF with an IP address. This means that the DF identity that needs authentication during an IKE exchange is the IP address. An IKE certificate for a DF contains the IP address of that DF. This IP address in the certificate MUST be used by IKE to validate the DF's IP address – to prevent IP address spoofing attacks.

After a pair of DF-DF SAs has been idle for some period of time, a DF MAY decide to remove it. In this case, the DF MUST send an ISAKMP Delete message to the other DF – to notify the other side of the SA deletion. Upon receiving a Delete message, the other DF MUST also remove that pair of SAs.

It will still be possible (with very small probability) that a DF uses a IPsec SA to send an event message to another DF; but when the event message arrives the target DF has already deleted the corresponding SA and has to drop the message. If there is still a problem after several timeouts and retries (e.g., ISAKMP Delete message was lost in transit), the sending DF MUST remove all of the corresponding IPsec SAs and re-run IKE to set up new SAs.

7.8.4.4 INTERFACE BETWEEN MGC AND DF

MGC and the DF MUST negotiate a pair of IPsec SAs (inbound and outbound) using IKE with pre-shared keys.

IKE will be running asynchronous to the event message generation and will guarantee that there is always a valid, non-expired pair of SAs.

At the DF, MGC Element IDs MUST somehow be associated with the corresponding IP addresses. One possibility is to associate each pre-shared key directly with the Element ID. IKE negotiations will use an ISAKMP identity payload of type ID_KEY_ID to identify the pre-shared key. The value in that identity payload will be the Element ID used in event messages.

Later, when an event message arrives at the DF, it will be able to query the database of SAs and retrieve a source IP address, based on the Element ID. The DF will make sure that it is the same as the source IP address in the IP packet header. One way to query this database is through SNMP, using an IPsec MIB.

7.8.5 Electronic Surveillance Security Profile Matrix

Table 29. Security Profile Matrix – Electronic Surveillance

	CMS-DF Events, MGC-DF Events & CMTS-DF Events	DF-DF Events	CMTS-DF Content & MG-DF Content	DF-DF Content
Authentication	yes	yes	yes (indirect)	yes (indirect)
Access control	yes	yes	optional	optional
Integrity	yes	yes	optional	optional
Confidentiality	yes	yes	yes	yes
Non-repudiation	no	no	no	no
Security mechanisms	IPsec with encryption and message integrity enabled Key management is IKE or Kerberos	IPsec with encryption and message integrity enabled Key management is IKE with certificates	RTP packets are already encrypted and authenticated by the source (MTA or MG)	RTP packets are already encrypted and authenticated by the source (MTA or MG)

7.9 CMS Provisioning

7.9.1 Reference Architecture

Provisioning is defined as the operations necessary to provide a specified service to a customer. PacketCable service provisioning can be viewed as two distinct operations: MTA provisioning and CMS subscriber provisioning. CMS provisioning refers to the interface between the Provisioning Server and the CMS.

7.9.2 Security Services

Authentication: Provisioning Server needs to be authenticated to prevent a third party from masquerading as a provisioning server to enable services for unauthorized MTAs. CMS needs to be authenticated to prevent someone from impersonating the CMS to receiving provisioning messages, thereby compromising privacy and deny service to provisioned MTAs.

Access Control: required along with authentication to prevent unauthorized access to provisioning data as well as denial-of-service.

Integrity: must be assured to disallow tampering with provisioning messages, in order to prevent a class of denial-of-service attacks.

Confidentiality: Provisioning messages contains private customer information, thus confidentiality is required.

7.9.3 Cryptographic Mechanisms

IPsec ESP MUST be used to both authenticate and encrypt the messages from CMS to Provisioning Server and vice versa. Refer to Section 6.1.2 for details of how IPsec ESP is used within PacketCable and for the list of available ciphersuites.

7.9.4 Key Management

Key management for the CMS-Provisioning Server interface is either IKE or Kerberos. Implementations MUST support IKE with pre-shared keys. Implementations MAY support IKE with X.509 certificates and they MAY support Kerberos using symmetric keys. For more information on the PacketCable use of IKE, refer to Section 6.2.2. For more information on the PacketCable use of Kerberos with symmetric keys, refer to Sections 6.4.3 and 6.5.

7.9.5 Provisioning Server-CMS Summary Security Profile Matrix

Table 30. **Security Profile Matrix – CMS Provisioning**

	CMS - Provisioning Server
Authentication	yes
Access control	yes
Integrity	yes
Confidentiality	yes
Non-repudiation	no
Security Mechanisms	IPsec IKE or Kerberos

8 PACKETCABLE CERTIFICATES

PacketCable uses digital certificates, which comply with the X.509 specification [33] and the IETF PKIX specification [34].

8.1 Generic Structure

8.1.1 Version

The Version of the certificates MUST be V3. All certificates MUST comply with [34] except where the non-compliance with the RFC is explicitly stated in this chapter of this document.

8.1.2 Public Key Type

RSA Public Keys are used throughout the hierarchy. The subjectPublicKeyInfo.algorithm.algorithm Object Identifier (OID) used MUST be 1.2.840.113549.1.1.1 (rsaEncryption).

The public exponent for all RSA PacketCable keys MUST be F4 - 65537.

8.1.3 Extensions

The following four extensions MUST be used as specified in the sections below. Any other certificate extensions MAY also be included but MUST be marked as non-critical.

8.1.3.1 subjectKeyIdentifier

The subjectKeyIdentifier extension included in all PacketCable CA certificates as required by [34] (e.g., all certificates except the device and ancillary certificates) MUST include the keyIdentifier value composed of the 160-bit SHA1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length and number of unused bits from the ASN1 encoding) (see [34]).

8.1.3.2 authorityKeyIdentifier

The authorityKeyIdentifier extension MUST be included in all PacketCable certificates, with the exception of root certificates, and MUST include a keyIdentifier value that is identical to the subjectKeyIdentifier in the CA certificate.

8.1.3.3 KeyUsage

The KeyUsage extension MUST be used for all PacketCable CA certificates and MUST be marked as critical with a value of keyCertSign and cRLSign. A KeyUsage extension MAY be included in end-entity certificates and SHOULD be marked as critical if included as specified in [34].

8.1.3.4 BasicConstraints

The basicConstraints extension MUST be used for all PacketCable CA certificates and MUST be marked as critical. The values for each certificate for basicConstraints MUST be marked as specified in each of the certificate description tables below.

8.1.4 Signature Algorithm

The signature mechanism used MUST be SHA-1 with RSA Encryption. The specific OID is 1.2.840.113549.1.1.5.

8.1.5 SubjectName and IssuerName

If a string cannot be encoded as a PrintableString it MUST be encoded as a UTF8String (tag [UNIVERSAL 12]).

When encoding an X.500 Name:

1. Each RelativeDistinguishedName (RDN) MUST contain only a single element in the set of X.500 attributes.
2. The order of the RDNs in an X.500 name MUST be the same as the order in which they are presented in this specification.

It should be noted that [34] and X.509 defines constraints (i.e. upper bounds) on the length of the attribute values. For example, the maximum length for common name (CN), organization name (O) and organizational unit (OU) name values is 64 characters. Where this specification mandates the inclusion of a static string in one of these values, (i.e. CN=<Company> PacketCable System Operator CA) companies MUST ensure that the addition of their identifying information does not cause the total length of the value to exceed the upper bound. In the case where a company's name causes the length of the value to exceed the upper bound, the vendor MUST truncate or abbreviate their information to ensure the total length does not exceed the upper bound.

8.1.6 Certificate Profile Notation

The tables below use the following notation:

- Extension details are specified by - [c:critical, n:non-critical; m:mandatory, o:optional].
- Optional subject naming attributes are surrounded by square brackets (e.g., [L = <city>]).
- Variable naming attribute values are surrounded by angle brackets. (e.g., CN = <Company Name> PacketCable CA). Values not surrounded by angle brackets are static and cannot be modified.

8.2 Certificate Trust Hierarchy

There are two distinct certificate hierarchies used in PacketCable.

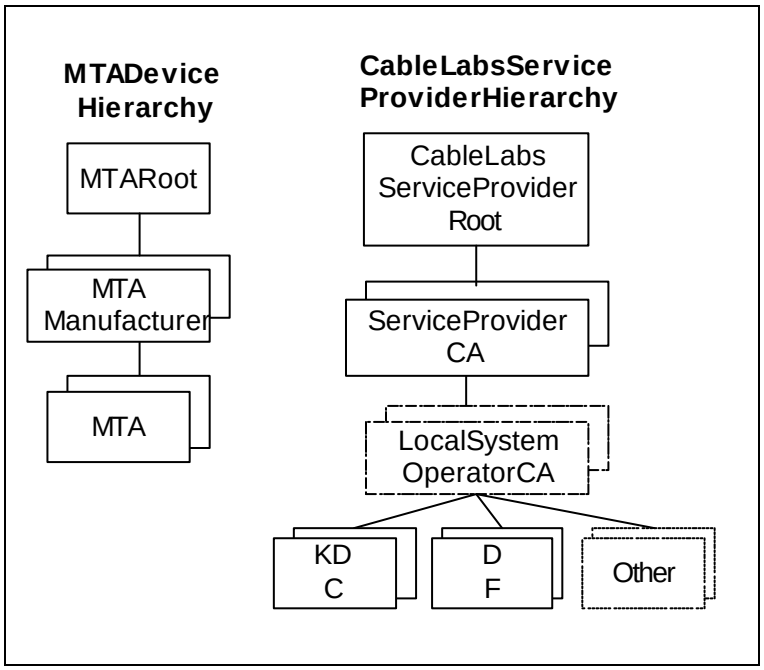


Figure 23. PacketCable Certificate Hierarchy

8.2.1 Certificate Validation

Within PacketCable certificate validation in general involves validation of a whole chain of certificates. As an example, when the Provisioning Server validates an MTA Device certificate, the actual chain of three certificates is validated:

MTA Root Certificate + MTA Manufacturer Certificate + MTA Device Certificate

The signature on the MTA Manufacturer Certificate is verified with the MTA Root Certificate and the signature on the MTA Device Certificate is verified with the MTA Manufacturer Certificate. The MTA Root Certificate is self-signed and is known in advance to the Provisioning Server. The public key present in the MTA Root Certificate is used to validate the signature on this same certificate.

Usually the first certificate in the chain is not explicitly included in the certificate chain that is sent over the wire. In the cases where the first certificate is explicitly included it MUST already be known to the verifying party ahead of time and MUST NOT contain any changes to the certificate with the possible exception of the certificate serial number, validity period and the value of the signature. If changes other than the certificate serial number, validity period and the value of the signature, exist in the CableLabs Service Provider Root certificate that was passed over the wire in comparison to the known CableLabs Service Provider Root certificate, the device making the comparison MUST fail the certificate verification.

The exact rules for certificate chain validation must fully comply with [34], where they are referred to as "Certificate Path Validation". In general, X.509 certificates support a liberal set of rules for determining if the issuer name of a certificate matches the subject name of another. The rules are such that two name fields may be declared to match even though a binary comparison of the two name fields does not indicate a match. [34] recommends that certificate authorities restrict the encoding of name fields so that an implementation can declare a match or mismatch using simple binary comparison. PacketCable security follows this recommendation. Accordingly, the DER-encoded tbsCertificate.issuer field of a PacketCable certificate MUST be an exact match to the DER-encoded tbsCertificate.subject field of its issuer certificate. An implementation MAY compare an issuer name to a subject name by performing a binary comparison of the DER-encoded tbsCertificate.issuer and tbsCertificate.subject fields.

The sections below specify the required certificate chain, which must be used to verify each certificate that appears at the leaf node (i.e., at the bottom) in the PacketCable certificate trust hierarchy illustrated in the above diagram.

Validity period nesting is not checked and intentionally not enforced. Thus, the validity period of a certificate need not fall within the validity period of the certificate that issued it.

8.2.2 MTA Device Certificate Hierarchy

The device certificate hierarchy exactly mirrors that of the DOCSIS1.1/BPI+ hierarchy. It is rooted at a CableLabs issued PacketCable MTA Root certificate, which is used as the issuing certificate of a set of manufacturer's certificates. The manufacturer's certificates are used to sign the individual device certificates.

The information contained in the following tables contains the PacketCable specific values for the required fields according to [34]. These PacketCable specific values MUST be followed according to the table below, except that Validity Periods SHOULD be as given in the tables. If a required field is not specifically listed for PacketCable then the guidelines in [34] MUST be followed.

8.2.2.1 MTA Root Certificate

This certificate MUST be verified as part of a certificate chain containing the MTA Root Certificate, MTA Manufacturer Certificate and the MTA Device Certificate.

Table 31. MTA Root Certificate

MTA Root Certificate	
Subject Name Form	C=US O=CableLabs OU=PacketCable CN=PacketCable Root Device Certificate Authority
Intended Usage	This certificate is used to sign MTA Manufacturer Certificates and is used by the KDC. This certificate is not used by the MTAs and thus does not appear in the MTA MIB.
Signed By	Self-Signed
Validity Period	20+ Years. It is intended that the validity period is long enough that this certificate is never re-issued.
Modulus Length	2048
Extensions	keyUsage[c,m](keyCertSign, cRLSign) subjectKeyIdentifier[n,m] basicConstraints[c,m](cA=true, pathLenConstraint=1)

8.2.2.2 MTA Manufacturer Certificate

This certificate MUST be verified as part of a certificate chain containing the MTA Root Certificate, MTA Manufacturer Certificate and the MTA Device Certificate.

The state/province, city and manufacturer's facility are optional attributes. A manufacturer may have more than one manufacturer's certificate, and there may exist one or more certificates per manufacturer. All Certificates for the same manufacturer may be provided to each MTA either at manufacture time or during a field update. The MTA MUST select an appropriate certificate for its use by matching the issuer name in the MTA Device Certificate with the subject name in the MTA Manufacturer Certificate. If present, the authorityKeyIdentifier of the device certificate MUST be matched to the subjectKeyIdentifier of the manufacturer certificate as described in [34].

The <CompanyName> field that is present in O and CN MAY be different in the two instances.

Table 32. MTA Manufacturer Certificate

MTA Manufacturer Certificate	
Subject Name Form	C=<country> O=<CompanyName> [ST=<state/province>] [L=<city>] OU=PacketCable [OU=<Manufacturer's Facility>] CN=<CompanyName> PacketCable CA
Intended Usage	This certificate is issued to each MTA manufacturer and can be provided to each MTA as part of the secure code download as specified by the PacketCable Security Specification (either at manufacture time, or during a field update). This certificate appears as a read-only parameter in the MTA MIB. This certificate along with the MTA Device Certificate is used to authenticate the MTA device identity (MAC address) during authentication by the KDC.
Signed By	MTA Root Certificate CA
Validity Period	20 Years
Modulus Length	2048
Extensions	keyUsage[c,m](keyCertSign, cRLSign) subjectKeyIdentifier[n,m] authorityKeyIdentifier[n,m](keyIdentifier=<subjectKeyIdentifier value from CA certificate>) basicConstraints[c,m](cA=true, pathLenConstraint=0)

8.2.2.3 MTA Device Certificate

This certificate MUST be verified as part of a certificate chain containing the MTA Root Certificate, MTA Manufacturer Certificate and the MTA Device Certificate.

The state/province, city and manufacturer's facility are optional attributes. The Manufacturer's Facility OU field, (if present) MAY be different from the Manufacturer's Facility OU field (if present) in the MTA Manufacturer certificate.

The MAC address MUST be expressed as six pairs of hexadecimal digits separated by colons, e.g., "00:60:21:A5:0A:23". The Alpha HEX characters (A-F) MUST be expressed as uppercase letters.

The MTA device certificate should not be replaced or renewed.

Table 33. MTA Device Certificate

MTA Device Certificate	
Subject Name Form	C=<country> O=<Company Name> [ST=<state/province>] [L=<city>] OU=PacketCable [OU=<Product Name>] [OU=<Manufacturer's Facility>] CN=<MAC Address>
Intended Usage	This certificate is issued by the MTA manufacturer and installed in the factory. The provisioning server cannot update this certificate. This certificate appears as a read-only parameter in the MTA MIB. This certificate is used to authenticate the MTA device identity (MAC address) during provisioning.
Signed By	MTA Manufacturer Certificate CA
Validity Period	At least 20 years
Modulus Length	1024, 1536 or 2048
Extensions	keyUsage[c,o](digitalSignature, keyEncipherment) authorityKeyIdentifier[n,m](keyIdentifier=<subjectKeyIdentifier value from CA certificate>)

8.2.3 CableLabs Service Provider Certificate Hierarchy

The Service Provider Certificate Hierarchy is rooted at a CableLabs issued CableLabs Service Provider Root certificate. That certificate is used as the issuing certificate of a set of service provider's certificates. The service provider's certificates are used to sign an optional local system certificate. If the local system certificate exists then that is used to sign the ancillary equipment certificates, otherwise the ancillary certificates are signed by the Service Provider's CA.

The information contained in the following table contains the specific values for the required fields according to [34]. These specific values MUST be followed according to the table below, except that Validity Periods SHOULD be as given in the tables. If a required field is not specifically listed then the guidelines in [34] MUST be followed.

8.2.3.1 CableLabs Service Provider Root Certificate

Before any Kerberos key management can be performed, an MTA and a KDC need to perform mutual authentication using the PKINIT extension to the Kerberos protocol. An MTA authenticates a KDC after it receives a PKINIT Reply message containing a KDC certificate chain. In authenticating the KDC, the MTA verifies the KDC certificate chain, including KDC's Service Provider Certificate signed by the CableLabs Service Provider Root CA.

Table 34. CableLabs Service Provider Root Certificate

CableLabs Service Provider Root Certificate	
Subject Name Form	C=US O=CableLabs CN=CableLabs Service Provider Root CA
Intended Usage	This certificate is used to sign Service Provider CA certificates. This certificate is installed into each MTA at the time of manufacture or with a secure code download as specified by the PacketCable Security Specification and cannot be updated by the Provisioning Server. Neither this root certificate nor the corresponding public key appears in the MTA MIB.
Signed By	Self-signed
Validity Period	20+. It is intended that the validity period is long enough that this certificate is never re-issued.
Modulus Length	2048
Extensions	keyUsage[c,m](keyCertSign, cRLSign) subjectKeyIdentifier[n,m] basicConstraints[c,m](cA=true)

8.2.3.2 Service Provider CA Certificate

This is the certificate held by the service provider, signed by the CableLabs Service Provider Root CA. It is verified as part of a certificate chain that includes the CableLabs Service Provider Root Certificate, Telephony Service Provider Certificate, optional Local System Certificate and an end-entity server certificate. The authenticating entities normally already possess the CableLabs Service Provider Root Certificate and it is not transmitted with the rest of the certificate chain.

The fact that a Service Provider CA Certificate is always explicitly included in the certificate chain allows a Service Provider the flexibility to change its certificate without requiring re-configuration of each entity that validates this certificate chain (e.g., MTA validating a PKINIT Reply). Each time the Service Provider CA Certificate changes, its signature MUST be verified with the CableLabs Service Provider Root Certificate. However, new certificate for the same Service Provider MUST preserve the same value of the OrganizationName attribute in the SubjectName.

The <Company> field that is present in O and CN MAY be different in the two instances.

Table 35. Service Provider CA Certificate

Service Provider CA Certificate	
Subject Name Form	C=<Country> O=<Company> CN=<Company> CableLabs Service Provider CA
Intended Usage	This certificate corresponds to a top-level Certification Authority within a domain of a single Service Provider. In order to make it easy to update this certificate, each network element is configured with the OrganizationName attribute of the Service Provider Certificate SubjectName. This is the only attribute in the certificate that must remain constant. In the case of an MTA, there is a read-write parameter in the MIB that identifies the OrganizationName attribute for each Kerberos realm (that may be shared among multiple MTA endpoints). The MTA does not accept Service Provider certificates that do not match this value of the OrganizationName attribute in the SubjectName. An MTA needs to perform the first PKINIT exchange with the MSO KDC right after a reboot, at which time its MIB tables are not yet configured. At that time, the MTA MUST accept any Service Provider OrganizationName attribute, but it MUST later check that the value added into the MIB for this realm is the same as the one in the initial PKINIT Reply.
Signed By	Signed by CableLabs Service Provider Certificate
Validity Period	20 years
Modulus Length	2048
Extensions	keyUsage[c,m](keyCertSign, cRLSign) subjectKeyIdentifier[n,m] authorityKeyIdentifier[n,m](keyIdentifier=<subjectKeyIdentifier value from CA certificate>) basicConstraints[c,m](cA=true, pathLenConstraint=1)

8.2.3.3 Local System CA Certificate

This is the certificate held by the local system. The existence of this certificate is optional, as the Service Provider CA may be used to directly sign all network server end-entity certificates. A certificate chain with a Local System CA Certificate MUST consist of the CableLabs Service Provider Root Certificate, Service Provider CA Certificate, Local System CA Certificate and an end-entity certificate.

The <Company> field that is present in O and CN MAY be different in the two instances.

Table 36. Local System CA Certificate

Local System CA Certificate	
Subject Name Form	C=<Country> O=<Company> OU=<Local System Name> CN=<Company> CableLabs Local System CA
Intended Usage	A Service Provider CA may delegate the issuance of certificates to a regional Certification Authority called Local System CA (with the corresponding Local System Certificate). Network servers are allowed to move freely between regional Certification Authorities of the same Service Provider. Therefore, the MTA MIB does not contain any information regarding a Local System Certificate (which might restrict an MTA to KDCs within a particular region).
Signed By	Service Provider CA Certificate
Validity Period	20 years.
Modulus Length	1024, 1536, 2048
Extensions	keyUsage[c,m](keyCertSign, cRLSign) subjectKeyIdentifier[n,m] authorityKeyIdentifier[n,m](keyIdentifier=<subjectKeyIdentifier value from CA certificate>) basicConstraints[c,m](cA=true, pathLenConstraint=0)

8.2.3.4 Operational Ancillary Certificates

All of these are signed by the either the Local System CA or by the Service Provider CA. Other ancillary certificates may be added to this standard at a later time.

8.2.3.4.1 Key Distribution Center Certificate

This certificate MUST be verified as part of a certificate chain containing the CableLabs Service Provider Root Certificate, Service Provider CA Certificate and the Ancillary Device Certificates.

The PKINIT specification in Appendix C requires the KDC certificate to include the subjectAltName v.3 certificate extension, the value of which must be the Kerberos principal name of the KDC.

Table 37. Key Distribution Center Certificate

Key Distribution Center Certificate	
Subject Name Form	C=<Country> O=<Company> [OU=<Local System Name>] OU= CableLabs Key Distribution Center CN=<DNS Name>
Intended Usage	To authenticate the identity of the KDC server to the MTA during PKINIT exchanges. This certificate is passed to the MTA inside the PKINIT replies and is therefore not included in the MTA MIB and cannot be updated or queried by the Provisioning Server.
Signed By	Service Provider CA Certificate or Local System Certificate
Validity Period	20 years.
Modulus Length	1024, 1536 or 2048
Extensions	keyUsage[c,o](digitalSignature) authorityKeyIdentifier[n,m](keyIdentifier=<subjectKeyIdentifier value from CA certificate>) subjectAltName[n,m](See Appendix C)

8.2.3.4.2 Delivery Function (DF)

This certificate MUST be verified as part of a certificate chain containing the CableLabs Service Provider Root Certificate, Service Provider CA Certificate and the Ancillary Device Certificates.

This certificate is used to sign phase 1 IKE intra-domain exchanges between DFs (which are used in Electronic Surveillance). Although Local System Name is optional, it is REQUIRED when the Local System CA signs this certificate. The IP address MUST be specified in standard dotted-quad notation, e.g., 245.120.75.22.

Table 38. DF Certificate

DF Certificate	
Subject Name Form	C=<Country> O=<Company> [OU=<Local System Name>] OU=PacketCable Electronic Surveillance CN=<IP address>
Intended Usage	To authenticate IKE key management, used to establish IPsec Security Associations between pairs of DFs. These Security Associations are used when a subject that is being legally wiretapped forwards the call and event messages containing call info have to be forwarded to a new wiretap server (DF).
Signed By	Service Provider CA Certificate or Local System CA Certificate
Validity Period	20 years
Modulus Length	2048
Extensions	keyUsage[c,o](digitalSignature) authorityKeyIdentifier[n,m](keyIdentifier=<subjectKeyIdentifier value from CA certificate>) subjectAltName[n,m](dNSName=<DNSName>)

8.2.3.4.3 PacketCable Server Certificates

These certificates MUST be verified as part of a certificate chain containing the CableLabs Service Provider Root Certificate, Service Provider Certificate, Local System Operator Certificate (if used) and the Ancillary Device Certificates.

These certificates are used to identify various servers in the PacketCable system. For example, they may be used to sign phase 1 IKE exchanges or to authenticate a PKINIT exchange. Although the Local System Name is optional, it is REQUIRED when the Local System CA signs this certificate. 2IP address values MUST be specified in standard dotted decimal notation: *e.g.*, 245.120.75.22. DNS Name values MUST be specified as a fully qualified domain name (FQDN): *e.g.*, device.packetcable.com.

Table 39. PacketCable Server Certificates

PacketCable Server Certificates	
Subject Name Form	C=<Country> O=<Company> OU=PacketCable OU=[<Local System Name>] OU=<Sub-System Name>[&<Sub-System Name>] CN=[<Server Identifier>] Or, CN=[<Element ID>][&<Element ID>] The CN will contain either a <Server Identifier> or one or more <Element ID>s. If the CN contains a <Server Identifier>, the value of <Server Identifier> MUST be the server's FQDN or its IP address, optionally followed by a colon (:) and an Element ID with no white space either before or after the colon. <Element ID> is the identifier that appears in billing event messages and it MUST be included in a certificate of every server that is capable of generating event messages. This includes a CMS, CMTS and MGC. There MAY be multiple <Element ID> fields, each separated by the character "&". [6] defines the Element ID as an 5-octet right-justified, space-padded ASCII-encoded numerical string. When converting the Element ID for use in a certificate, any spaces MUST be converted to ASCII zeroes (0x30). For example, a CMTS that has the Element ID " 311" will have a common name "00311". The value of <Sub-System Name> MUST be one of the following: • For Border Proxy: bp • For Cable Modem Termination System: cmts • For Call Management Server: cms • For Media Gateway: mg • For Media Gateway Controller: mgc • For Media Player: mp • For Media Player Controller: mpc • For Provisioning Server: ps • For Record Keeping Server: rks • For Signaling Gateway: sg Components that contain combined elements (such as a CMS with an integrated MGC) MUST indicate this in the Subject Name by including all Sub-System Names, joined with the character "&", in the OU field. In the case of combined elements, a single Element ID or multiple Element IDs may be used. If multiple Element IDs are used, all Element IDs MUST be included in the CN, and the order of these Element IDs MUST correspond to the order of the Sub-System Name fields in the OU. The following is an example OU and CN for a combined CMS and MGC. The CMS with Element ID " 311" and a MGC with Element ID " 312". OU=cms&mgc CN=00311&00312 The following is an example OU and CN for a combined CMS and MGC. In this case, the CMS and MGC share a single Element ID of " 311". OU=cms&mgc CN=00311&00311

Intended Usage	These certificates are used to identify various servers in the PacketCable system. For example they may be used to sign phase 1 IKE exchanges or to authenticate a device in a PKINIT exchange
Signed By	Telephony Service Provider Certificate or Local System Certificate
Validity Period	Set by MSO policy
Modulus Length	2048
Extensions	keyUsage[c,o](digitalSignature, keyEncipherment) authorityKeyIdentifier[n,m](keyIdentifier=<subjectKeyIdentifier value from CA cert>) subjectAltName[n,o](dNSName=<DNSName> | iPAddress=<IP Address Name>) The keyUsage tag is optional. When it is used it MUST be marked as critical. The subjectAltName extension MUST be included for all servers that are capable of generating event messages. For all other servers, the subjectAltName extension MAY be included. If the subjectAltName extension is included, it MUST include the corresponding name value as specified in the CN field of the subject.

8.2.3.4.4 TLS Certificates

These certificates MUST be verified as part of a certificate chain containing the CableLabs Service Provider Root Certificate, Service Provider Certificate, Local System Operator Certificate (if used) and the Ancillary Device Certificates.

These certificates are used to authenticate TLS handshake exchanges (and encrypt when using RSA key exchange). Although the Local System Name is optional, it is REQUIRED when the Local System CA signs this certificate. DNS Name values MUST be specified as a fully qualified domain name (FQDN): e.g., device.packetcable.com.

Table 40. PacketCable TLS Certificates

PacketCable Server Certificates	
Subject Name Form	C=<Country> O=<Company> OU=[<Local System Name>] OU=PacketCable CN=[<Server Identifier>] The value of <Server Identifier> MUST be the server's FQDN. Note that only a single FQDN can be included in the CN field.
Intended Usage	These certificates are used to authenticate TLS handshake exchanges (and encrypt when using RSA key exchange).
Signed By	Telephony Service Provider Certificate or Local System Certificate
Validity Period	Set by MSO policy
Modulus Length	1024, 1536, 2048
Extensions	KeyUsage[c,m](digitalSignature, keyEncipherment) extendedKeyUsage[n,m] (id-kp-serverAuth, id-kp-clientAuth) authorityKeyIdentifier[n,m](keyIdentifier=<subjectKeyIdentifier value from CA cert>)

8.2.4 Certificate Revocation

Out of scope for PacketCable at this time.

9 CRYPTOGRAPHIC ALGORITHMS

This section describes the cryptographic algorithms used in the PacketCable security specification. When a particular algorithm is used, the algorithm MUST follow the corresponding specification.

9.1 AES

AES-128 is a 128-bit block cipher that MUST be implemented according to the AES (Advanced Encryption Standard) proposed submission specified in [35]. AES-128 is used in CBC mode with a 128-bit block size in PacketCable. AES-128 requires 10 rounds of cryptographic operations in encryption or decryption. The Initialization Vector for CBC mode is specified for each use of AES in PacketCable.

In 1997, the National Institute of Standards and Technology (NIST) initiated a process to select a symmetric-key encryption algorithm to be used to protect sensitive (unclassified) Federal information in furtherance of NIST's statutory responsibilities. In 1998, NIST announced the acceptance of fifteen candidate algorithms and requested the assistance of the cryptographic research community in analyzing the candidates. This analysis included an initial examination of the security and efficiency characteristics for each algorithm. NIST reviewed the results of this preliminary research and selected MARS, RC6(tm), Rijndael, Serpent and Twofish as finalists. Having reviewed further public analysis of the finalists, NIST has decided to propose Rijndael as the Advanced Encryption Standard.

9.2 DES

The Data Encryption Standard (DES) is specified in [31] For Media Stream encryption, PacketCable does not require error checking on the DES key, and the full 64-bits of key provided to the DES algorithm will be generated according to Section 7.6.2.3.3.1.

9.2.1 XDESX

An option for the encryption of RTP packets is DESX-XEX, XDESX, or DESX, has been proven as a viable method for overcoming the weaknesses in DES while not greatly adding to the implementation complexity. The strength of DESX against key search attacks is presented in [45]. The CBC mode of DESX-XEX is shown a figure below, where DESX-XEX is executed within the block called "block cipher." Inside the block, DESX-XEX is performed as shown in a figure below using a 192-bit key. K1 is the first 8-bytes of the key, and K2 represents the second 8-bytes of key; and K3 the third 8-bytes of key.

9.2.2 DES-CBC-PAD

This variant of DES is also based on the analysis of DESX presented in [45]. When using DESX in CBC mode, an optimized architecture is possible. It can be described in terms of the DES-CBC configuration plus the application of a random pad on the final DES-CBC output blocks. This configuration uses 128-bits of keying material, where 64-bits are applied to the DES block according to [31], and an additional 64-bits of keying material is applied as the random pad on the final DES-CBC output blocks.

In this case, the same IV used to initialize the CBC mode is used as keying material for the random pad. Each block of DES-CBC encrypted output is XOR-ed with the 64-bit Initialization Vector that was used to start the CBC operation. If a short block results from using Residual Block Termination (see Section 9.3), the left-most-bits of the IV are used in the final XOR padding operation. This mode of DES-CBC is shown a figure below, where DES is executed in the block called "block cipher." A 64-bit key value is used.

9.2.3 3DES-EDE

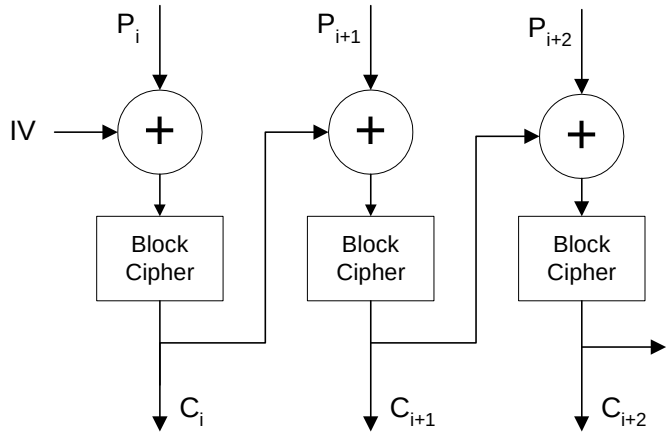
Another option for the encryption of RTP packets for PacketCable, is 3DES-EDE-CBC. The CBC mode of 3DES-EDE is shown in a figure below, where 3DES-EDE is executed within the block called "block cipher." Inside the block, 3DES-EDE is performed as shown in a figure below using a 128-bit key. K1 is the first 8-bytes of the key, and K2 represents the second 8-bytes of key; and K3=K1.

9.3 Block Termination

If block ciphers are supported, a short block (n bits < block size depending on the cipher algorithms) MUST be terminated by residual block termination as shown in the figure below. Residual block termination (RBT) is executed as follows:

Given a final block having n bits, where n is less than block size, the n bits are padded up to a block by appending (block size – n) bites of arbitrary value to the right of the n-bits. The resulting block is encrypted using B-bit CFB mode, with the next-to-last ciphertext block serving as the initialization vector for the CFB operation (see [43], B. Schneier's Applied Cryptography). Here, B stands for the cipher-specific block size. The leftmost n bits of the resulting ciphertext are used as the short cipher block. In the special case where the complete payload is less than the cipher block size, the procedure is the same as for a short final block, with the provided initialization vector serving as the initialization vector for the operation. Residual block termination is illustrated in the figure below for both encryption and decryption operations.

CBC Encryption Architecture



CBC Decryption Architecture

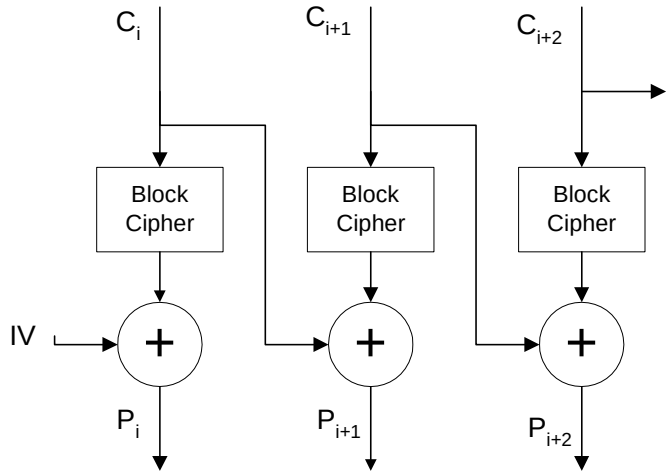
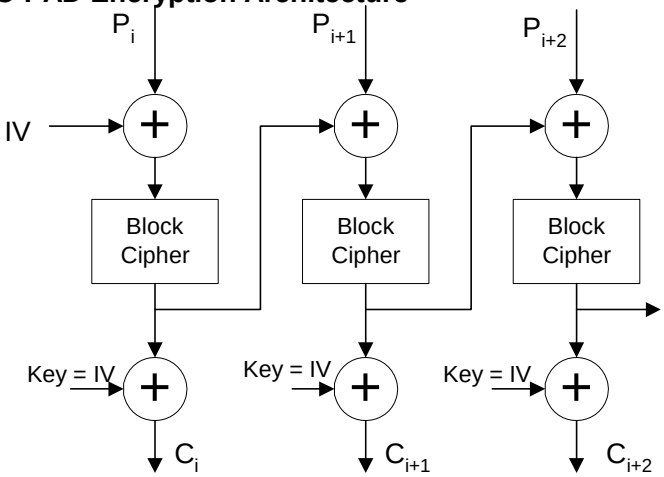


Figure 24. CBC Mode

CBC-PAD Encryption Architecture



CBC-PAD Decryption Architecture

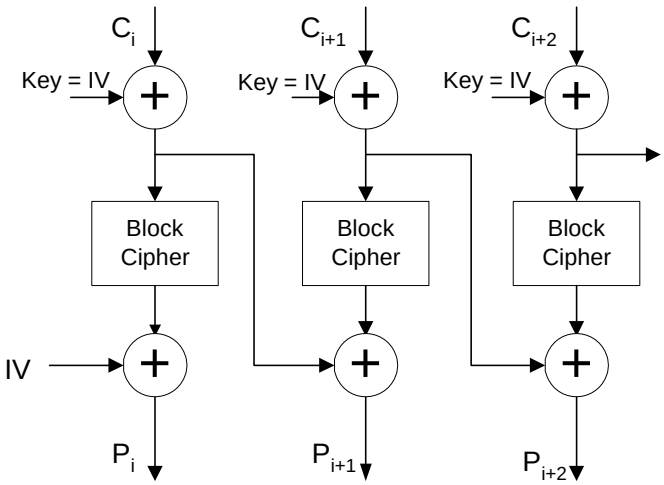


Figure 25. CBC Pad Mode

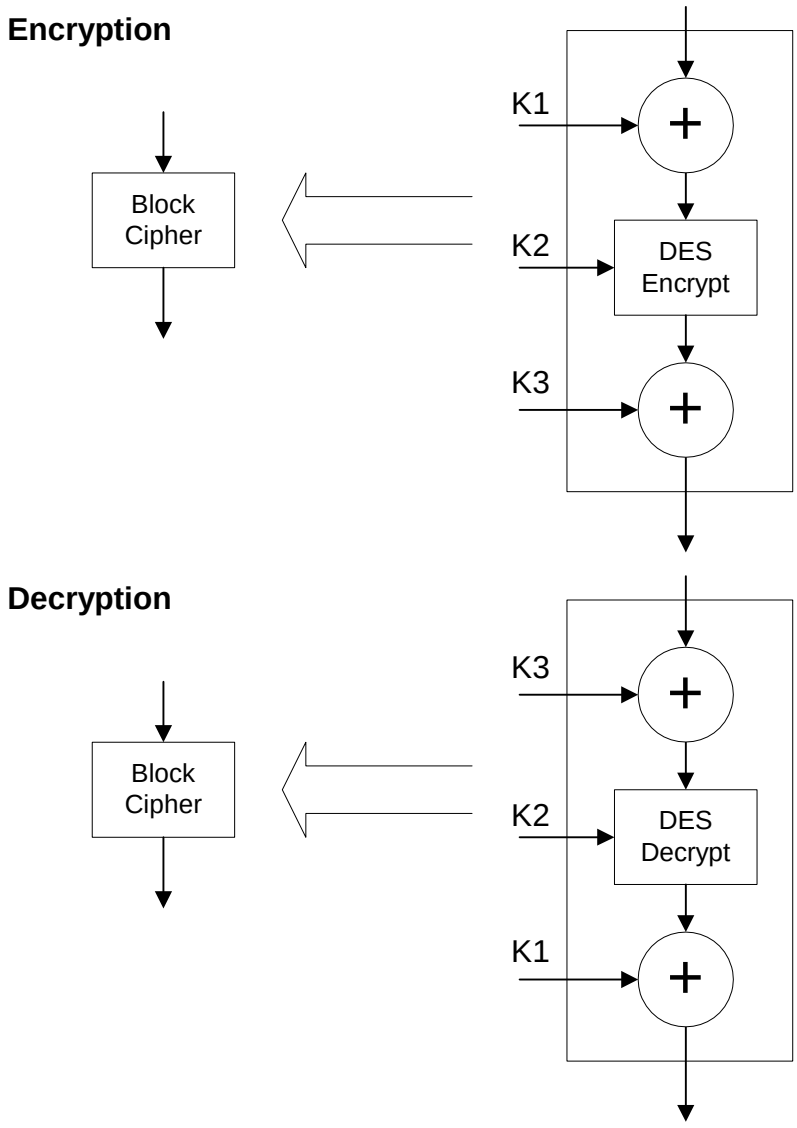


Figure 26. DESX-XEX as Block Cipher

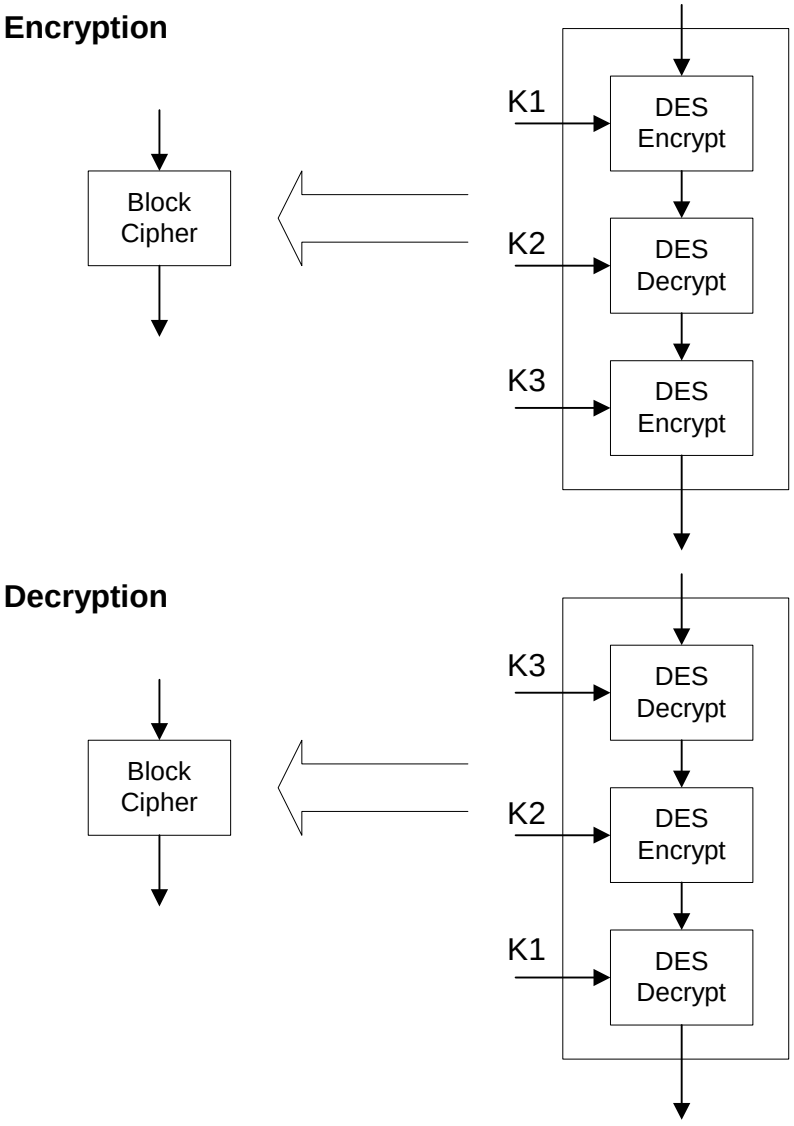
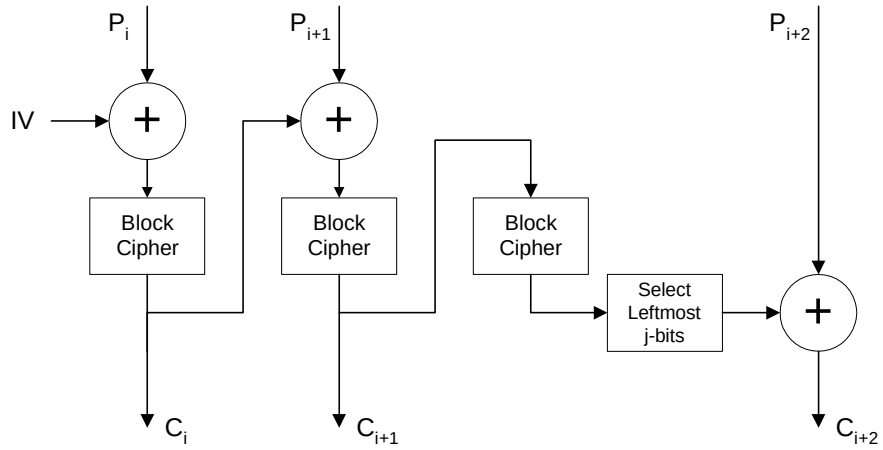


Figure 27. 3DES-EDE as Block Cipher

Encryption
CBC w/ Residual Block Termination



Decryption
CBC w/ Residual Block Termination

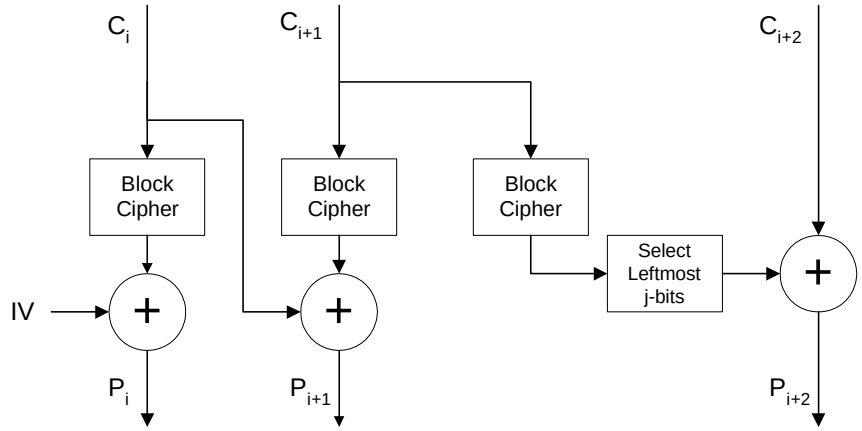


Figure 28. CBC with Residual Block Termination

9.4 RSA Signature

All public key signatures for PacketCable MUST be generated and verified using the RSA signature algorithm described in [16] The format for all PacketCable signatures MUST be compliant with the Cryptographic Message Syntax [12].

9.5 HMAC-SHA1

The keyed hash employed by the HMAC-Digest Attribute MUST use the HMAC message authentication method [11] with the SHA-1 hash algorithm [15].

9.6 Key Derivation

Key derivation sections in this document refer to a function F(S, seed), where S is a shared secret from which keying material is derived, and seed is a constant string of bytes. Below is the specification of F(S, seed), borrowed from TLS [17]:

F(S, seed) = HMAC_SHA-1(S, A(1) + seed) +
HMAC_SHA-1(S, A(2) + seed) +
HMAC_SHA-1(S, A(3) + seed) + …

where + indicates concatenation.

A() is defined as: A(0) = seed

A(i) = HMAC_SHA-1(S, A(i-1))

F(S, seed) is iterated as many times as is necessary to produce required quantity of data. Unused bytes at the end of the last iteration will be discarded.

9.7 The MMH-MAC

In this section the MMH Function and the MMH Message Authentication Code (MAC) are described. The MMH-MAC is the message authentication code option for the media flows. As discussed in Section 7.6.2, the MMH-MAC is computed over the RTP header and the payload is generated by the codec. The MMH Function will be described next, followed by a description of the MMH-MAC.

9.7.1 The MMH Function

The Multilinear Modular Hash (MMH) Function described below is a variant of the MMH Function described in [18]. Some of the computations described below use signed arithmetic whereas the computations in [18] use unsigned arithmetic. The signed arithmetic variant described here was selected for its computational efficiency when implemented on DSPs. All of the properties shown for the MMH function in [18] continue to hold for the signed variant.

The MMH Function has three parameters: the word size, the number of words of input, and the number of words of output. MMH[ω,s,t] specifies the hash function with word size ω, s input words and t output words. For PacketCable the word size is fixed to 16 bits: ω=16. The number of output words will be either 1 or 2: $t \in \{1,2\}$. The MMH Hash Function will first be described for t=1, i.e., one output word.

9.7.1.1 MMH[16,s,1]

For the remainder of this Section 9.7, MMH[16,s,1] is denoted by H. In addition to s words of input, H also takes as input a key of s words. When H is used in computing the MMH-MAC, the key is randomly generated and remains fixed for several inputs as described in Section 9.7.2. The key is denoted by k and the ith word of the key by ki: k=k1,k2,…,ks. Likewise the input message is denoted by m and the ith word of the input message by mi: m = m1, m2,…, ms.

To describe *H*, the following definitions are needed. For any even positive integer n, S_n is defined to be the following set of n integers: {-n/2,…,0,…,(n/2)-1}. For example, $S_{2^{16}}$ = {-2^{15},…,0,…,2^{15}-1} is the set of

signed 16 bit integers. For any integer z, z smod n is the unique element ω of S_n such that $z \equiv \omega \pmod{n}$. For example, if z is a 32 bit signed integer in 32 bit twos complement representation, then z smod 2^{16} can be computed by taking the 16 least significant bits of z and interpreting those bits in 16 bit twos complement representation.

For any positive integer q, Z_q denotes the following set of q integers: {0, 1, …, q-1}.

As described above H takes as input a key of s words. Each of the s words is interpreted as a 16 bit signed integer, i.e., an element of $S_{2^{16}}$. H also takes as input a message of s words. Each of the s words is interpreted as a 16 bit signed integer, i.e., an element of $S_{2^{16}}$. The output of H is an unsigned 16-bit integer, i.e., an element of $Z_{2^{16}}$. Alternatively, the range of H is $S_{2^{16}}^s \times S_{2^{16}}^s$ and the domain is $Z_{2^{16}}$.

H is defined by a series of steps. For $k,m \in S_{2^{16}}^s$,

1. Define H_1 as $H_1(k,m) = \sum_{i=1}^{s} k_i \cdot m_i \; s \bmod 2^{32}$.
2. Define H_2 as $H_2(k,m) = H_1(k,m) \bmod p$ where p is the prime number $p = 2^{16}+1$.
3. Define H as $H(k,m) = H_2(k,m) \bmod 2^{16}$.

Equivalently,

$$H(k,m) = \left(\left(\left(\sum_{i=1}^{s} k_i \cdot m_i \right) s \bmod 2^{32} \right) \bmod p \right) \bmod 2^{16}$$

Each step is discussed in detail below.

Step1. $H_1(k,m)$ is the inner product of two vectors each of s 16 bit signed integers. The result of the inner product is taken smod 2^{32} to yield an element of $S_{2^{32}}$.[5] That is, if the inner product is in twos complement representation of 32 or more bits, the 32 least significant bits are retained and the resulting integer is interpreted in 32 bit twos complement representation.

Step 2. This step consists of taking an element x of $S_{2^{32}}$ and reducing it mod p to yield an element of Z_p. If x is represented in 32 bit twos complement notation then this reduction can be accomplished very simply as follows. Let a be the unsigned integer given by the 16 most significant bits of x. Let b be the unsigned integer given by the 16 least significant bits of x. There are two cases depending upon whether x is negative.

Case 1. If x is non-negative then $x = a2^{16}+b$ where $a \in \{0,\ldots,2^{15}-1\}$ and $b \in \{0,\ldots,2^{16}-1\}$. From the modular equation

$$a2^{16}+b \equiv a2^{16} + b - a(2^{16}+1) \pmod{(2^{16}+1)}$$

it follows that $x \equiv b - a \pmod{p}$. The quantity b-a is in the range $\{-2^{15}+1,\ldots,2^{16}-1\}$. Therefore if b-a is non-negative then $x \bmod p = b - a$. If $b - a$ is negative then $x \bmod p = b$-a+p.

Case 2. If x is negative then $x = a2^{16} + b - 2^{32}$ where $a \in \{2^{15},\ldots,2^{16}-1\}$ and $b \in \{0,\ldots,2^{16}-1\}$. From the modular equation

$$a2^{16} + b - 2^{32} \equiv b + a2^{16} - a(2^{16}+1) - 2^{32} + 2^{16}(2^{16}+1) \pmod{(2^{16}+1)}$$

it follows that $x \equiv b - a + 2^{16} \pmod{p}$. The range of the quantity $b - a + 2^{16}$ is given by:

$$1 \le b - a + 2^{16} \le 2^{17} - 2^{15} - 1 \le 2p - 1$$

[5] The entire sum need not be computed before performing the smod 2^{32} operation. The smod 2^{32} operation can be computed on partial sums since (x + y) smod 2^{32} = (x smod 2^{32} + y smod 2^{32}) smod 2^{32}.

Therefore, if $b - a + 2^{16} < p$ then $x \bmod p = b - a + 2^{16}$. If $b - a + 2^{16} \geq p$ then $x \bmod p = b - a + 2^{16} - p$.

Step 3. This step takes an element of Z_p and reduces it mod 2^{16}. This is equivalent to taking the 16 least significant bits.

9.7.1.2 MMH[16,s,2]

This section describes the MMH Function with an output length of two words, which in this case is 32 bits. For convenience, let *H'* = MMH[16,*s*,2]. *H'* takes a key of *s*+1 words. Let $k = k_1,\ldots,k_{s+1}$. Furthermore, define $k^{(1)}$ to be the *s* words of *k* starting with k_1, i.e., $k^{(1)} = k_1,\ldots,k_s$. Define $k^{(2)}$ to be the *s* words of *k*, starting with k_2, i.e., $k^{(2)} = k_2,\ldots,k_{s+1}$. For any $k \in S_{2^{16}}^{s+1}$ and any $m \in S_{2^{16}}^{s}$ *m*, *H'*(*k*,*m*) is computed by first computing $H(k^{(1)},m)$ and then $H(k^{(2)},m)$ and concatenating the results. That is, $H'(k,m) = H(k^{(1)},m) \circ H(k^{(2)},m)$.

9.7.2 The MMH-MAC

This section describes the MMH-MAC. The MMH-MAC has three parameters; the word size, the number of words of input, and the number of words of output. MMH-MAC[ω,*s*,*t*] specifies the message authentication code with word size ω, *s* input words and *t* output words. For PacketCable the wordsize is fixed to 16 bits: ω = 16. The number of output words will be either 1 or 2: $t \in \{1,2\}$.

For convenience, let *M* = MMH-MAC[16,*s*,*t*]. When using *M*, a sender and receiver share a key *k* of *s* + *t* - *1* words. In addition, they share a sequence of key streams of *t* words each, one one-time pad for each message sent. Let $r^{(i)}$ be the key stream used for the *i*th message sent and received. For the *i*th message, $m^{(i)}$, the message authentication code is computed as:

$$M(k, r^{(i)}, m^{(i)}) = \mathrm{H}(k, m^{(i)}) + r^{(i)} .$$

Here *H* = MMH[16,*s*,*t*], $r^{(i)}$ is in $Z_{2^{16}}$ and addition is mod 2^{16}

9.7.2.1 MMH-MAC When Using a Block Cipher

When calculating the MMH-MAC when encryption is performed by one of the available block ciphers, the block cipher is used to calculate the t words of r (i) key stream (pad) as defined in Section 7.6.2.1.2.2.3.

9.7.2.2 Handling Variable-Size Data

In order to handle data of all possible sizes up to a maximum value, the following rules MUST be followed for computing an MMH function:

- If the data is not a multiple of the word size, pad the data up to a multiple of the word size (16-bits) with zero-bytes. In other words, if the length of message *m* is not a multiple of word size w, but rather of length b octets, b = n * w + r with n >= 0 and 0 < r < w, then pad message m at the end with w–r zero-bytes before passing it as the input to *M*.
- It the key is larger than what is needed for a particular message, truncate the key. In other words, if a message *m* is not of length *s* words, but rather of length *v* < s words, then truncate the value of the key k to *v*+t-1 words before it is used to calculate the MMH hash. (For MMH hash with 1 word output, t=1 and k is truncated to *v* words. For 2 word output, t=2 and k is truncated to *v*+1 words.)

9.8 Random Number Generation

Good random number generation is vital to most cryptographic mechanisms. Implementations SHOULD do their best to produce true-random seeds; they should also use cryptographically strong pseudo-random number generation algorithms. RFC 1750 (See [44]) gives some suggestions; other possibilities include use of a per-MTA secret installed at manufacture time and used in the random number generation process.

10 PHYSICAL SECURITY

10.1 Protection for MTA Key Storage

An MTA MUST maintain in permanent write-once memory an RSA key pair. An MTA SHOULD deter unauthorized physical access to this keying material.

The level of physical protection of keying material required by the PacketCable security specification for an MTA is specified in terms of the security levels defined in the FIPS PUBS 140-2, Security Requirements for Cryptographic Modules, standard (see [46]). An MTA SHOULD, at a minimum meet FIPS PUBS 140-2 Security Level 1 requirements.

The PacketCable Security specification's minimal physical security requirements for an MTA will not, in normal practice, jeopardize a customer's data privacy. Assuming the subscriber controls the access to the MTA with the same diligence they would protect a cellular phone, physical attacks on that MTA to extract keying data are likely to be detected by the subscriber.

An MTA's weak physical security requirements, however, could undermine the cryptographic protocol's ability to meet its main security objective: to provide a service operator with strong protection from theft of high value networks.

The PacketCable Security specification requirements protect against unauthorized access to these network services by enforcing an end-to-end message integrity and encryption of signaling flows across the network and by employing an authenticated key management protocol. If an attacker is able to legitimately subscribe to a set of services and also gain physical access to an MTA containing keying material, then in the absence of strong physical protection of this information, the attacker can extract keying material from the MTA, and redistribute the keys to other users running modified illegitimate MTA's, effectively allowing theft of network services.

There are two distinct variations of "active attacks" involving the extraction and redistribution of cryptographic keys. These include the following:

1. An "RSA active clone" would actively participate in PacketCable key exchanges. An attacker must have some means by which to remove the cryptographic keys that enable services, from the clone master, and install these keys into a clone MTA. An active clone would work in conjunction with an active clone master to passively obtain the clone master's keying material and then actively impersonate the clone master. A single active clone may have numerous active clone master identities from which to select to obtain access to network services. This attack allows, for example, the theft of non-local voice communications.

2. An DH active clone would also actively participate in the PacketCable key exchanges and like the RSA active clone, would require an attacker to extract the cryptographic keys that enable the service from the clone master and install these keys into a clone MTA. However, unlike the RSA active clone, the DH active clone must obtain the clone masters random number through alternate means or perform the key exchange and risk detection. Like an RSA active clone, an DH active clone may have numerous clone master identities from which to select to obtain access to the network services.

3. An "active black box" MTA, holding another MTA's session or IPsec keys, would use the keys to obtain access to network-based services or traffic flows similar to the RSA active clone. Since both session keys and IPsec keys change frequently, such clones have to be periodically updated with the new keying material, using some out-of-band means.

   An active RSA clone, for example, could operate on a cable access network within whatever geographic region the cloned parent MTA was authorized to operate in. Depending upon the degree to which a service operator's subscriber authorization system restricted the location from which the MTA could operate, the clone's scope of operation could extend well beyond a single DOCSIS MAC domain.

An active clone attack may be detectable by implementing the appropriate network controls in the system infrastructure. Depending on the access fraud detection methods that are in place, a service operator has a good probability of detecting a clone's operation should it attempt to operate within the network. The service operator could then take defensive measures against the detected clone. For example, in the case of an active RSA clone, it could block the device's future network access by including the device certificate on the certificate hot list. Also the service operator's subscriber authorization system could limit the geographic region over which a subscriber, identified by its cryptographic credentials, could operate. Additionally the edge router functionality in the CMTS could limit any access based upon IP address. These methods would limit the region over which an active RSA clone could operate and reduce the financial incentive for such an attack.

The architectural guidelines for PacketCable security are determined by balancing the revenues that could be lost due to the classes of active attacks against the cost of the methods to prevent the attack. At the extreme side of preventive methods available to thwart attacks, both physical security equivalent to FIPS PUB 140-2 Level 3 and network based fraud detection methods could be used to limit the access fraud that allows theft of network based services. The network based intrusion detection of active attacks allows operators to consider operational defenses as an alternative to increased physical security. If the revenues threatened by the active attacks increase significantly to the point where additional protective mechanisms are necessary, the long term costs of operational defenses would need to be compared with the costs of migrating to MTAs with stronger physical security. The inclusion of physical security should be an implementation and product differentiation specific decision.

Although the scope of the current PacketCable specifications do not specifically define requirements for MTAs to support any requirements other than voice communications, the goal of the PacketCable effort is to provide for the eventual inclusion of integrated services. Part of these integrated services may include the "multicast" of high value content or extremely secure multicast corporate videoconference sessions.

Two additional attacks enabling a compromise of these types of services are defined:

1. An "RSA passive clone" passively monitors the parent MTA's key exchanges and, having a copy of the parent MTA's RSA private key, is able to obtain the same traffic keying material the parent MTA has access to. The clone then uses the keying material to decrypt downstream traffic flows it receives across the shared medium. This attack is limited in that it only allows snooping, but if the traffic were of high value, the attack could facilitate the theft of high value multicast traffic.

2. A "Passive black box" MTA, holding another MTA's short-term (relative to the RSA key) keys, uses the keying material to gain access to encrypted traffic flows similar to the RSA passive clone.

   The passive attacks, unlike the active attacks, are not detectable using network based intrusion detection techniques since these units never make themselves known to the network while performing the attack. However, this type of service theft has unlimited scale since the passive clones and black boxes, even though they operate on different cable access networks (sometimes referred to as the same DOCSIS MAC domain) as the parent MTA from whom the keys were extracted, gain access to the protected data the parent MTA is currently receiving since the encryption of the data most likely occurred at the source. (These are general IP multicast services, not to be confused with the specific DOCSIS 1.1 / BPI+ multicast implementation, where passive clones would be restricted to a single downstream CMTS segment.) The snooping of the point-to-point data is limited to the DOCSIS MAC domain of the parent MTA. Passive attacks may be prevented by ensuring that the cryptographic keys that are used to enable the services cannot be tampered with in any manner.

In setting goals and guidelines for the PacketCable security architecture, an assessment has to be made of the value of the services and content that can be stolen or monitored by key extraction and redistribution to passive MTAs. The cost of the solution should not be greater that the lost revenue due to theft of the service or subscribers terminating the service due to lack of privacy. However at this time, there is no clear cost that can be attributed to either the lost revenue from high value multicast services or the loss of subscribers due to privacy issues unique to this type of network. Therefore, it was concluded that passive key extraction and redistribution attacks would pose an indeterminate financial risk to service operators; and that the cost of protection (i.e., incorporation of stronger physical security into the MTA) should be balanced against the value of the risk. As with the active attacks, the decision to include additional functionality to implement physical security in the MTA should be left as an implementation and product differentiation issue and not be mandated as a requirement of the PacketCable security specification.

10.2 MTA Key Encapsulation

As stated in the previous section, FIPS PUB 140-2 Security Level 1 specifies very little actual physical security and that an MTA MUST deter unauthorized "physical" access to its keying material. This restricted access also includes any ability to directly read the keying material using any of the MTA interfaces.

One of the (many) requirements of FIPS PUB 140-2 Security Level 3 is that "the entry or output of plaintext Critical Security Parameters (CSPs) be performed using ports that are physically separated from other ports, or interfaces that are logically separated using a trusted path from other interfaces. Plaintext CSPs may be entered into or output from the cryptographic module in encrypted form (in which case they may travel through enclosing or intervening systems)". As also mentioned in the previous section, the PacketCable security specification is not requiring compliance with any of the FIPS PUB 140-2 Security Level 3 requirements.

However, it is strongly recommended that any persistent keying material SHOULD be encapsulated such that there is no way to extract the keying material from the MTA using any of the MTA interfaces (either required in the PacketCable specifications or proprietary provided by the vendor) without modifications to the MTA.

In particular, an MTA subscriber may also be connected to the Internet via a Cable Modem (which may be embedded in the same MTA). In that case, hackers may potentially exploit any weakness in the configuration of the subscriber’s local network and steal MTA’s secret and private keys over the network. If instead, the MTA subscriber is connected to a company Intranet, the same threat still exists, although from a smaller group of people.

11 SECURE SOFTWARE DOWNLOAD

PacketCable 1.5 includes only Embedded MTAs. E-MTAs are embedded with DOCSIS 1.1 cable modems (including BPI+). E-MTAs MUST have their software upgraded by the Cable Modem according to the DOCSIS 1.1 requirements as specified in [8] and [9].

Appendix A. PacketCable Admin Guidelines & Best Practices (Informative)

This section describes various administration guidelines and best practices recommended by PacketCable. These are included to help facilitate network administration and/or strengthen overall security in the PacketCable network.

A.1 Routine CMS Service Key Refresh

PacketCable recommends that the CMS service keys be routinely changed (refreshed) at least once every 90 days in order to reduce the risk of key compromises. The refresh period should be a provisioned parameter that can be use in one the following ways:

In the case of manual key changes, an administrator is prompted or reminded to manually change a CMS service key.

In the case of autonomous key changes (using Kerberos Set/Change Password) it will define the refresh period.

Note that in the case of autonomous key refreshes, whereby administrative overhead and scalability are not an issue, it may be desirable to use a refresh period that is less than 90 days (but at least the maximum ticket lifetime). This may further reduce the risk of key compromise.

Appendix B. Kerberos Network Authentication Service (Normative)

The Kerberos Network Authentication Service specification is currently still an IETF draft. This document complies only with the version of the draft that is included in this section. The PacketCable security team will continue to track progress of the Kerberos Network Authentication Service draft through the IETF.

The Kerberos Network Authentication Service

INTERNET-DRAFT

Clifford Neuman
John Kohl
Theodore Ts'o
November 24, 2000
Expires May 24, 2001

The Kerberos Network Authentication Service (V5)

draft-ietf-cat-kerberos-revisions-07.txt.

STATUS OF THIS MEMO

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC 2026. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.ietf.org (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

The distribution of this memo is unlimited. It is filed as draft-ietf-cat-kerberos-revisions-07.txt, and expires May 24, 2001. Please send comments to: ietf-krb-wg@anl.gov

ABSTRACT

This document provides an overview and specification of Version 5 of the Kerberos protocol, and updates RFC1510 to clarify aspects of the protocol and its intended use that require more detailed or clearer explanation than was provided in RFC1510. This document is intended to provide a detailed description of the protocol, suitable for implementation, together with descriptions of the appropriate use of protocol messages and fields within those messages.

This document is not intended to describe Kerberos to the end user, system administrator, or application developer. Higher level papers describing Version 5 of the Kerberos system [NT94] and documenting version 4 [SNS88], are available elsewhere.

OVERVIEW

This INTERNET-DRAFT describes the concepts and model upon which the Kerberos network authentication system is based. It also specifies Version 5 of the Kerberos protocol.

The motivations, goals, assumptions, and rationale behind most design decisions are treated cursorily; they are more fully described in a paper available in IEEE communications [NT94] and earlier in the Kerberos portion of the Athena Technical Plan [MNSS87]. The protocols have been a proposed standard and are being considered for advancement for draft standard through the IETF standard process. Comments are encouraged on the presentation, but only minor refinements to the protocol as implemented or extensions that fit within current protocol framework will be considered at this time.

Requests for addition to an electronic mailing list for discussion of Kerberos, kerberos@MIT.EDU, may be addressed to kerberos-request@MIT.EDU. This mailing list is gatewayed onto the Usenet as the group comp.protocols.kerberos. Requests for further information, including documents and code availability, may be sent to info-kerberos@MIT.EDU.

BACKGROUND

The Kerberos model is based in part on Needham and Schroeder's trusted third-party authentication protocol [NS78] and on modifications suggested by Denning and Sacco [DS81]. The original design and implementation of Kerberos Versions 1 through 4 was the work of two former Project Athena staff members, Steve Miller of Digital Equipment Corporation and Clifford Neuman (now at the Information Sciences Institute of the University of Southern California), along with Jerome Saltzer, Technical Director of Project Athena, and Jeffrey Schiller, MIT Campus Network Manager. Many other members of Project Athena have also contributed to the work on Kerberos.

Version 5 of the Kerberos protocol (described in this document) has evolved from Version 4 based on new requirements and desires for features not available in Version 4. The design of Version 5 of the Kerberos protocol was led by Clifford Neuman and John Kohl with much input from the community. The development of the MIT reference implementation was led at MIT by John Kohl and Theodore T'so, with help and contributed code from many others. Since RFC1510 was issued, extensions and revisions to the protocol have been proposed by many individuals. Some of these proposals are reflected in this document. Where such changes involved significant effort, the document cites the contribution of the proposer.

Reference implementations of both version 4 and version 5 of Kerberos are publicly available and commercial implementations have been developed and are widely used. Details on the differences between Kerberos Versions 4 and 5 can be found in [KNT92].

1. Introduction

Kerberos provides a means of verifying the identities of principals, (e.g. a workstation user or a network server) on an open (unprotected) network. This is accomplished without relying on assertions by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets traveling along the network can be read, modified, and inserted at will[1.1]. Kerberos performs authentication under these conditions as a trusted third-party authentication service by using conventional (shared secret key [1.2]) cryptography. Kerberos extensions described in [PKINIT reference as RFC] provide for the use of public key cryptography during certain phases of the authentication protocol. These extensions allow

authentication of users registered with public key certification authorities, and provide certain benefits of public key cryptography in situations where they are needed.

The basic Kerberos authentication process proceeds as follows: A client sends a request to the authentication server (AS) requesting 'credentials' for a given server. The AS responds with these credentials, encrypted in the client's key. The credentials consist of 1) a 'ticket' for the server and 2) a temporary encryption key (often called a "session key"). The client transmits the ticket (which contains the client's identity and a copy of the session key, all encrypted in the server's key) to the server. The session key (now shared by the client and server) is used to authenticate the client, and may optionally be used to authenticate the server. It may also be used to encrypt further communication between the two parties or to exchange a separate sub-session key to be used to encrypt further communication.

Implementation of the basic protocol consists of one or more authentication servers running on physically secure hosts. The authentication servers maintain a database of principals (i.e., users and servers) and their secret keys. Code libraries provide encryption and implement the Kerberos protocol. In order to add authentication to its transactions, a typical network application adds one or two calls to the Kerberos library directly or through the Generic Security Services Application Programming Interface, GSSAPI, described in separate document [ref to GSSAPI RFC]. These calls result in the transmission of the necessary messages to achieve authentication.

The Kerberos protocol consists of several sub-protocols (or exchanges). There are two basic methods by which a client can ask a Kerberos server for credentials. In the first approach, the client sends a cleartext request for a ticket for the desired server to the AS. The reply is sent encrypted in the client's secret key. Usually this request is for a ticket-granting ticket (TGT) which can later be used with the ticket-granting server (TGS). In the second method, the client sends a request to the TGS. The client uses the TGT to authenticate itself to the TGS in the same manner as if it were contacting any other application server that requires Kerberos authentication. The reply is encrypted in the session key from the TGT. Though the protocol specification describes the AS and the TGS as separate servers, they are implemented in practice as different protocol entry points within a single Kerberos server.

Once obtained, credentials may be used to verify the identity of the principals in a transaction, to ensure the integrity of messages exchanged between them, or to preserve privacy of the messages. The application is free to choose whatever protection may be necessary.

To verify the identities of the principals in a transaction, the client transmits the ticket to the application server. Since the ticket is sent "in the clear" (parts of it are encrypted, but this encryption doesn't thwart replay) and might be intercepted and reused by an attacker, additional information is sent to prove that the message originated with the principal to whom the ticket was issued. This information (called the authenticator) is encrypted in the session key, and includes a timestamp. The timestamp proves that the message was recently generated and is not a replay. Encrypting the authenticator in the session key proves that it was generated by a party possessing the session key. Since no one except the requesting principal and the server know the session key (it is never sent over the network in the clear) this guarantees the identity of the client.

The integrity of the messages exchanged between principals can also be guaranteed using the session key (passed in the ticket and contained in the

credentials). This approach provides detection of both replay attacks and message stream modification attacks. It is accomplished by generating and transmitting a collision-proof checksum (elsewhere called a hash or digest function) of the client's message, keyed with the session key. Privacy and integrity of the messages exchanged between principals can be secured by encrypting the data to be passed using the session key contained in the ticket or the sub-session key found in the authenticator.

The authentication exchanges mentioned above require read-only access to the Kerberos database. Sometimes, however, the entries in the database must be modified, such as when adding new principals or changing a principal's key. This is done using a protocol between a client and a third Kerberos server, the Kerberos Administration Server (KADM). There is also a protocol for maintaining multiple copies of the Kerberos database. Neither of these protocols are described in this document.

1.1. Cross-realm operation

The Kerberos protocol is designed to operate across organizational boundaries. A client in one organization can be authenticated to a server in another. Each organization wishing to run a Kerberos server establishes its own 'realm'. The name of the realm in which a client is registered is part of the client's name, and can be used by the end-service to decide whether to honor a request.

By establishing 'inter-realm' keys, the administrators of two realms can allow a client authenticated in the local realm to prove its identity to servers in other realms[1.3]. The exchange of inter-realm keys (a separate key may be used for each direction) registers the ticket-granting service of each realm as a principal in the other realm. A client is then able to obtain a ticket-granting ticket for the remote realm's ticket-granting service from its local realm. When that ticket-granting ticket is used, the remote ticket-granting service uses the inter-realm key (which usually differs from its own normal TGS key) to decrypt the ticket-granting ticket, and is thus certain that it was issued by the client's own TGS. Tickets issued by the remote ticket-granting service will indicate to the end-service that the client was authenticated from another realm.

A realm is said to communicate with another realm if the two realms share an inter-realm key, or if the local realm shares an inter-realm key with an intermediate realm that communicates with the remote realm. An authentication path is the sequence of intermediate realms that are transited in communicating from one realm to another.

Realms are typically organized hierarchically. Each realm shares a key with its parent and a different key with each child. If an inter-realm key is not directly shared by two realms, the hierarchical organization allows an authentication path to be easily constructed. If a hierarchical organization is not used, it may be necessary to consult a database in order to construct an authentication path between realms.

Although realms are typically hierarchical, intermediate realms may be bypassed to achieve cross-realm authentication through alternate authentication paths (these might be established to make communication between two realms more efficient). It is important for the end-service to know which realms were transited when deciding how much faith to place in the authentication process. To facilitate this decision, a field in each ticket contains the names of the realms that were involved in authenticating the client.

The application server is ultimately responsible for accepting or rejecting authentication and should check the transited field. The application server

may choose to rely on the KDC for the application server's realm to check the transited field. The application server's KDC will set the TRANSITED-POLICY-CHECKED flag in this case. The KDC's for intermediate realms may also check the transited field as they issue ticket-granting-tickets for other realms, but they are encouraged not to do so. A client may request that the KDC's not check the transited field by setting the DISABLE-TRANSITED-CHECK flag. KDC's are encouraged but not required to honor this flag.

1.2. Choosing a principal with which to communicate

The Kerberos protocol provides the means for verifying (subject to the assumptions in 1.4) that the entity with which one communicates is the same entity that was registered with the KDC using the claimed identity (principal name). It is still necessary to determine whether that identity corresponds to the entity with which one intends to communicate.

When appropriate data has been exchanged in advance, this determination may be performed syntactically by the application based on the application protocol specification, information provided by the user, and configuration files. For example, the server principal name (including realm) for a telnet server might be derived from the user specified host name (from the telnet command line), the "host/" prefix specified in the application protocol specification, and a mapping to a Kerberos realm derived syntactically from the domain part of the specified hostname and information from the local Kerberos realms database.

One can also rely on trusted third parties to make this determination, but only when the data obtained from the third party is suitably integrity protected wile resident on the third party server and when transmitted. Thus, for example, one should not rely on an unprotected domain name system record to map a host alias to the primary name of a server, accepting the primary name as the party one intends to contact since an attacker can modify the mapping and impersonate the party with which one intended to communicate.

If a Kerberos server supports name canonicalization, it may be relied upon as a third party to aid in this determination. When utilizing the name canonicalization function provided by the Kerberos server, a client, having already located the instance of a service it wishes to contact, makes a request to the KDC using the server's name information as specified by the user. The Kerberos server will attempt to locate a service principal in its database that corresponds to the requested name and return a ticket for the appropriate server principal to the client. If the KDC determines that the correct server principal is registered in another realm, the KDC will provide a referral to the Kerberos realm that is known to contain the requested service principal. The name canonicalization function supports identity mapping only, and it may not be used as a general name service to locate service instances. There is no guarantee that the returned server principal name (identity) will embed the name of the host on which the server resides.

1.3. Authorization

As an authentication service, Kerberos provides a means of verifying the identity of principals on a network. Authentication is usually useful primarily as a first step in the process of authorization, determining whether a client may use a service, which objects the client is allowed to access, and the type of access allowed for each. Kerberos does not, by itself, provide authorization. Possession of a client ticket for a service provides only for authentication of the client to that service, and in the absence of a separate authorization procedure, it should not be considered

by an application as authorizing the use of that service.

Such separate authorization methods may be implemented as application specific access control functions and may utilize files on the application server, or on separately issued authorization credentials such as those based on proxies [Neu93], or on other authorization services. Separately authenticated authorization credentials may be embedded in a tickets authorization data when encapsulated by the kdc-issued authorization data element.

Applications should not accept the mere issuance of a service ticket by the Kerberos server (even by a modified Kerberos server) as granting authority to use the service, since such applications may become vulnerable to the bypass of this authorization check in an environment if they interoperate with other KDCs or where other options for application authentication (e.g. the PKTAPP proposal) are provided.

1.4. Environmental assumptions

Kerberos imposes a few assumptions on the environment in which it can properly function:

* 'Denial of service' attacks are not solved with Kerberos. There are places in the protocols where an intruder can prevent an application from participating in the proper authentication steps. Detection and solution of such attacks (some of which can appear to be not-uncommon 'normal' failure modes for the system) is usually best left to the human administrators and users.
* Principals must keep their secret keys secret. If an intruder somehow steals a principal's key, it will be able to masquerade as that principal or impersonate any server to the legitimate principal.

* 'Password guessing' attacks are not solved by Kerberos. If a user chooses a poor password, it is possible for an attacker to successfully mount an offline dictionary attack by repeatedly attempting to decrypt, with successive entries from a dictionary, messages obtained which are encrypted under a key derived from the user's password.
* Each host on the network must have a clock which is 'loosely synchronized' to the time of the other hosts; this synchronization is used to reduce the bookkeeping needs of application servers when they do replay detection. The degree of "looseness" can be configured on a per-server basis, but is typically on the order of 5 minutes. If the clocks are synchronized over the network, the clock synchronization protocol must itself be secured from network attackers.
* Principal identifiers are not recycled on a short-term basis. A typical mode of access control will use access control lists (ACLs) to grant permissions to particular principals. If a stale ACL entry remains for a deleted principal and the principal identifier is reused, the new principal will inherit rights specified in the stale ACL entry. By not re-using principal identifiers, the danger of inadvertent access is removed.

1.5. Glossary of terms

Below is a list of terms used throughout this document.

Authentication
: Verifying the claimed identity of a principal.

Authentication header
: A record containing a Ticket and an Authenticator to be presented to a server as part of the authentication process.

Authentication path

A sequence of intermediate realms transited in the authentication process when communicating from one realm to another.

Authenticator
A record containing information that can be shown to have been recently generated using the session key known only by the client and server.

Authorization
The process of determining whether a client may use a service, which objects the client is allowed to access, and the type of access allowed for each.

Capability
A token that grants the bearer permission to access an object or service. In Kerberos, this might be a ticket whose use is restricted by the contents of the authorization data field, but which lists no network addresses, together with the session key necessary to use the ticket.

Ciphertext
The output of an encryption function. Encryption transforms plaintext into ciphertext.

Client
A process that makes use of a network service on behalf of a user. Note that in some cases a Server may itself be a client of some other server (e.g. a print server may be a client of a file server).

Credentials
A ticket plus the secret session key necessary to successfully use that ticket in an authentication exchange.

KDC
Key Distribution Center, a network service that supplies tickets and temporary session keys; or an instance of that service or the host on which it runs. The KDC services both initial ticket and ticket-granting ticket requests. The initial ticket portion is sometimes referred to as the Authentication Server (or service). The ticket-granting ticket portion is sometimes referred to as the ticket-granting server (or service).

Kerberos
Aside from the 3-headed dog guarding Hades, the name given to Project Athena's authentication service, the protocol used by that service, or the code used to implement the authentication service.

Plaintext
The input to an encryption function or the output of a decryption function. Decryption transforms ciphertext into plaintext.

Principal
A named client or server entity that participates in a network communication, with one name that is considered canonical.

Principal identifier
The canonical name used to uniquely identify each different principal.

Seal
To encipher a record containing several fields in such a way that the fields cannot be individually replaced without either knowledge of the encryption key or leaving evidence of tampering.

Secret key
An encryption key shared by a principal and the KDC, distributed outside the bounds of the system, with a long lifetime. In the case of a human user's principal, the secret key may be derived from a password.

Server
A particular Principal which provides a resource to network clients. The server is sometimes referred to as the Application Server.

Service
A resource provided to network clients; often provided by more than one server (for example, remote file service).

Session key
: A temporary encryption key used between two principals, with a lifetime limited to the duration of a single login "session".

Sub-session key
: A temporary encryption key used between two principals, selected and exchanged by the principals using the session key, and with a lifetime limited to the duration of a single association.

Ticket
: A record that helps a client authenticate itself to a server; it contains the client's identity, a session key, a timestamp, and other information, all sealed using the server's secret key. It only serves to authenticate a client when presented along with a fresh Authenticator.

2. Ticket flag uses and requests

Each Kerberos ticket contains a set of flags which are used to indicate attributes of that ticket. Most flags may be requested by a client when the ticket is obtained; some are automatically turned on and off by a Kerberos server as required. The following sections explain what the various flags mean, and gives examples of reasons to use such a flag.

2.1. Initial, pre-authenticated, and hardware authenticated tickets

The INITIAL flag indicates that a ticket was issued using the AS protocol and not issued based on a ticket-granting ticket. Application servers that want to require the demonstrated knowledge of a client's secret key (e.g. a password-changing program) can insist that this flag be set in any tickets they accept, and thus be assured that the client's key was recently presented to the application client.

The PRE-AUTHENT and HW-AUTHENT flags provide additional information about the initial authentication, regardless of whether the current ticket was issued directly (in which case INITIAL will also be set) or issued on the basis of a ticket-granting ticket (in which case the INITIAL flag is clear, but the PRE-AUTHENT and HW-AUTHENT flags are carried forward from the ticket-granting ticket).

2.2. Invalid tickets

The INVALID flag indicates that a ticket is invalid. Application servers must reject tickets which have this flag set. A postdated ticket will usually be issued in this form. Invalid tickets must be validated by the KDC before use, by presenting them to the KDC in a TGS request with the VALIDATE option specified. The KDC will only validate tickets after their starttime has passed. The validation is required so that postdated tickets which have been stolen before their starttime can be rendered permanently invalid (through a hot-list mechanism) (see section 3.3.3.1).

2.3. Renewable tickets

Applications may desire to hold tickets which can be valid for long periods of time. However, this can expose their credentials to potential theft for equally long periods, and those stolen credentials would be valid until the expiration time of the ticket(s). Simply using short-lived tickets and obtaining new ones periodically would require the client to have long-term access to its secret key, an even greater risk. Renewable tickets can be used to mitigate the consequences of theft. Renewable tickets have two "expiration times": the first is when the current instance of the ticket expires, and the second is the latest permissible value for an individual expiration time. An application client must periodically (i.e. before it expires) present a renewable ticket to the KDC, with the RENEW option set in

the KDC request. The KDC will issue a new ticket with a new session key and a later expiration time. All other fields of the ticket are left unmodified by the renewal process. When the latest permissible expiration time arrives, the ticket expires permanently. At each renewal, the KDC may consult a hot-list to determine if the ticket had been reported stolen since its last renewal; it will refuse to renew such stolen tickets, and thus the usable lifetime of stolen tickets is reduced.

The RENEWABLE flag in a ticket is normally only interpreted by the ticket-granting service (discussed below in section 3.3). It can usually be ignored by application servers. However, some particularly careful application servers may wish to disallow renewable tickets.

If a renewable ticket is not renewed by its expiration time, the KDC will not renew the ticket. The RENEWABLE flag is reset by default, but a client may request it be set by setting the RENEWABLE option in the KRB_AS_REQ message. If it is set, then the renew-till field in the ticket contains the time after which the ticket may not be renewed.

2.4. Postdated tickets

Applications may occasionally need to obtain tickets for use much later, e.g. a batch submission system would need tickets to be valid at the time the batch job is serviced. However, it is dangerous to hold valid tickets in a batch queue, since they will be on-line longer and more prone to theft. Postdated tickets provide a way to obtain these tickets from the KDC at job submission time, but to leave them "dormant" until they are activated and validated by a further request of the KDC. If a ticket theft were reported in the interim, the KDC would refuse to validate the ticket, and the thief would be foiled.

The MAY-POSTDATE flag in a ticket is normally only interpreted by the ticket-granting service. It can be ignored by application servers. This flag must be set in a ticket-granting ticket in order to issue a postdated ticket based on the presented ticket. It is reset by default; it may be requested by a client by setting the ALLOW-POSTDATE option in the KRB_AS_REQ message. This flag does not allow a client to obtain a postdated ticket-granting ticket; postdated ticket-granting tickets can only by obtained by requesting the postdating in the KRB_AS_REQ message. The life (endtime-starttime) of a postdated ticket will be the remaining life of the ticket-granting ticket at the time of the request, unless the RENEWABLE option is also set, in which case it can be the full life (endtime-starttime) of the ticket-granting ticket. The KDC may limit how far in the future a ticket may be postdated.

The POSTDATED flag indicates that a ticket has been postdated. The application server can check the authtime field in the ticket to see when the original authentication occurred. Some services may choose to reject postdated tickets, or they may only accept them within a certain period after the original authentication. When the KDC issues a POSTDATED ticket, it will also be marked as INVALID, so that the application client must present the ticket to the KDC to be validated before use.

2.5. Proxiable and proxy tickets

At times it may be necessary for a principal to allow a service to perform an operation on its behalf. The service must be able to take on the identity of the client, but only for a particular purpose. A principal can allow a service to take on the principal's identity for a particular purpose by granting it a proxy.

The process of granting a proxy using the proxy and proxiable flags is used to provide credentials for use with specific services. Though conceptually

also a proxy, user's wishing to delegate their identity for ANY purpose must use the ticket forwarding mechanism described in the next section to forward a ticket granting ticket.

The PROXIABLE flag in a ticket is normally only interpreted by the ticket-granting service. It can be ignored by application servers. When set, this flag tells the ticket-granting server that it is OK to issue a new ticket (but not a ticket-granting ticket) with a different network address based on this ticket. This flag is set if requested by the client on initial authentication. By default, the client will request that it be set when requesting a ticket granting ticket, and reset when requesting any other ticket.

This flag allows a client to pass a proxy to a server to perform a remote request on its behalf, e.g. a print service client can give the print server a proxy to access the client's files on a particular file server in order to satisfy a print request.

In order to complicate the use of stolen credentials, Kerberos tickets are usually valid from only those network addresses specifically included in the ticket[2.1]. When granting a proxy, the client must specify the new network address from which the proxy is to be used, or indicate that the proxy is to be issued for use from any address.

The PROXY flag is set in a ticket by the TGS when it issues a proxy ticket. Application servers may check this flag and at their option they may require additional authentication from the agent presenting the proxy in order to provide an audit trail.

2.6. Forwardable tickets

Authentication forwarding is an instance of a proxy where the service granted is complete use of the client's identity. An example where it might be used is when a user logs in to a remote system and wants authentication to work from that system as if the login were local.

The FORWARDABLE flag in a ticket is normally only interpreted by the ticket-granting service. It can be ignored by application servers. The FORWARDABLE flag has an interpretation similar to that of the PROXIABLE flag, except ticket-granting tickets may also be issued with different network addresses. This flag is reset by default, but users may request that it be set by setting the FORWARDABLE option in the AS request when they request their initial ticket-granting ticket.

This flag allows for authentication forwarding without requiring the user to enter a password again. If the flag is not set, then authentication forwarding is not permitted, but the same result can still be achieved if the user engages in the AS exchange specifying the requested network addresses and supplies a password.

The FORWARDED flag is set by the TGS when a client presents a ticket with the FORWARDABLE flag set and requests a forwarded ticket by specifying the FORWARDED KDC option and supplying a set of addresses for the new ticket. It is also set in all tickets issued based on tickets with the FORWARDED flag set. Application servers may choose to process FORWARDED tickets differently than non-FORWARDED tickets.

2.7 Transited Policy Checking

While the application server is ultimately responsible for accepting or rejecting authentication and should check the transited field, a KDC may apply a realm specific policy for validating the transited field and

accepting credentials for cross-realm authentication. When the KDC applies such checks and accepts such cross-realm authentication it will set the TRANSITED-POLICY-CHECKED flag in the service tickets it issues based on the cross-realm TGT. A client may request that the KDC's not check the transited field by setting the DISABLE-TRANSITED-CHECK flag. KDC's are encouraged but not required to honor this flag.

2.8 Anonymous Tickets

When policy allows, a KDC may issue anonymous tickets for the purpose of enabling encrypted communication between a client and server without identifying the client to the server. Such anonymous tickets are issued with a generic principal name configured on the KDC (e.g. "anonymous@") and will have the ANONYMOUS flag set. A server accepting such a ticket may assume that subsequent requests using the same ticket and session key originate from the same user. Requests with the same username but different tickets are likely to originate from different users. Users request anonymous ticket by setting the REQUEST-ANONYMOUS option in an AS or TGS request.

2.9. Other KDC options

There are three additional options which may be set in a client's request of the KDC.

2.9.1 Name canonicalization [JBrezak]

The NAME-CANONICALIZATION option allows the KDC to replace the name of the client or server requested by the client with the canonical form of the principal's name, if known, or to refer the client to a KDC for the realm with which the requested principal is registered.

Where name canonicalization is supported a client who can identify a principal but does not know the full principal name can request that the Kerberos server attempt to lookup the name in its database and use the canonical name of the requested principal or return a referral to a realm that has the requested principal in its namespace. Use of name canonicalization supports the case where a principal has multiple common names (names typed by a user[2.2]), all of which are known to the KDC, but only one Kerberos identity (the canonical name is the Kerberos principal name). Name canonicalization is intended solely to provide a secure mapping from the name known by a user to its principal identifier. It is not intended for use as a general purpose nameserver or to identify instances of a service.

The CANONICALIZE flag in a ticket request is used to indicate to the Kerberos server that the client will accept an alternative name to the principal in the request or a referral to another realm. When name canonicalization is supported in a realm, all instances of the AS and TGS for the realm must be able to interpret requests with this flag. In realms where name canonicalization is not supported, this flag may be ignored. By using this flag, the client can avoid extensive configuration needed to map specific host names to a particular realm.

2.9.2 Renewable-OK

The RENEWABLE-OK option indicates that the client will accept a renewable ticket if a ticket with the requested life cannot otherwise be provided. If a ticket with the requested life cannot be provided, then the KDC may issue a renewable ticket with a renew-till equal to the requested endtime. The value of the renew-till field may still be adjusted by site-determined limits or limits imposed by the individual principal or server.

2.9.3 ENC-TKT-IN-SKEY

The ENC-TKT-IN-SKEY option supports user-to-user authentication. It allows the KDC to issue a service ticket encrypted using the session key from a ticket granting ticket issued to another user. This is needed to support peer-to-peer authentication since the long term key of the user does not remain on the workstation after initial login. The ENC-TKT-IN-SKEY option is honored only by the ticket-granting service. It indicates that the ticket to be issued for the end server is to be encrypted in the session key from the additional second ticket-granting ticket provided with the request. See section 3.3.3 for specific details.

3. Message Exchanges

The following sections describe the interactions between network clients and servers and the messages involved in those exchanges.

3.1. The Authentication Service Exchange

Summary

Message direction	Message type	Section
1. Client to Kerberos	KRB_AS_REQ	5.4.1
2. Kerberos to client	KRB_AS_REP or KRB_ERROR	5.4.2 5.9.1

The Authentication Service (AS) Exchange between the client and the Kerberos Authentication Server is initiated by a client when it wishes to obtain authentication credentials for a given server but currently holds no credentials. In its basic form, the client's secret key is used for encryption and decryption. This exchange is typically used at the initiation of a login session to obtain credentials for a Ticket-Granting Server which will subsequently be used to obtain credentials for other servers (see section 3.3) without requiring further use of the client's secret key. This exchange is also used to request credentials for services which must not be mediated through the Ticket-Granting Service, but rather require a principal's secret key, such as the password-changing service[3.1]. This exchange does not by itself provide any assurance of the identity of the user[3.2].

The exchange consists of two messages: KRB_AS_REQ from the client to Kerberos, and KRB_AS_REP or KRB_ERROR in reply. The formats for these messages are described in sections 5.4.1, 5.4.2, and 5.9.1.

In the request, the client sends (in cleartext) its own identity and the identity of the server for which it is requesting credentials. The response, KRB_AS_REP, contains a ticket for the client to present to the server, and a session key that will be shared by the client and the server. The session key and additional information are encrypted in the client's secret key. The KRB_AS_REP message contains information which can be used to detect replays, and to associate it with the message to which it replies.

Without pre-authentication, the authentication server does not know whether the client is actually the principal named in the request. It simply sends a reply without knowing or caring whether they are the same. This is acceptable because nobody but the principal whose identity was given in the request will be able to use the reply. Its critical information is encrypted in that principal's key. The initial request supports an optional field that can be used to pass additional information that might be needed for the initial exchange. This field may be used for pre-authentication as described in section 3.1.1.

Various errors can occur; these are indicated by an error response (KRB_ERROR) instead of the KRB_AS_REP response. The error message is not encrypted. The KRB_ERROR message contains information which can be used to associate it with the message to which it replies. If suitable preauthentication has occurred, an optional checksum may be included in the KRB_ERROR message to prevent fabrication or modification of the KRB_ERROR message. When a checksum is not present, the lack of integrity protection precludes the ability to detect replays, fabrications, or modifications of the message, and the client must not depend on information in the KRB_ERROR message for security critical operations.

3.1.1. Generation of KRB_AS_REQ message

The client may specify a number of options in the initial request. Among these options are whether pre-authentication is to be performed; whether the requested ticket is to be renewable, proxiable, or forwardable; whether it should be postdated or allow postdating of derivative tickets; whether the client requests name-canonicalization or an anonymous ticket; and whether a renewable ticket will be accepted in lieu of a non-renewable ticket if the requested ticket expiration date cannot be satisfied by a non-renewable ticket (due to configuration constraints; see section 4). See section A.1 for pseudocode.

The client prepares the KRB_AS_REQ message and sends it to the KDC.

3.1.2. Receipt of KRB_AS_REQ message

If all goes well, processing the KRB_AS_REQ message will result in the creation of a ticket for the client to present to the server. The format for the ticket is described in section 5.3.1. The contents of the ticket are determined as follows.

3.1.3. Generation of KRB_AS_REP message

The authentication server looks up the client and server principals named in the KRB_AS_REQ in its database, extracting their respective keys. If the requested client principal named in the request is not known because it doesn't exist in the KDC's principal database and if an acceptable canonical name of the client is not known, then an error message with a KDC_ERR_C_PRINCIPAL_UNKNOWN is returned.

If the request had the CANONICALIZE option set and if the AS finds the canonical name for the client and it is in another realm, then an error message with a KDC_ERR_WRONG_REALM error code and the cname and crealm in the error message will contain the true client principal name and realm. In this case, since no key is shared with the client, the response from the KDC is not integrity protected and the referral can only be considered a hint; the validity of the referral is validated upon successful completion of initial authentication with the correct AS using the appropriate user key.

If required, the server pre-authenticates the request, and if the pre-authentication check fails, an error message with the code KDC_ERR_PREAUTH_FAILED is returned. If pre-authentication is required, but was not present in the request, an error message with the code KDC_ERR_PREAUTH_FAILED is returned and the PA-ETYPE-INFO pre-authentication field will be included in the KRB-ERROR message. If the server cannot accommodate an encryption type requested by the client, an error message with code KDC_ERR_ETYPE_NOSUPP is returned. Otherwise the KDC generates a 'random' session key[3.3].

When responding to an AS request, if there are multiple encryption keys registered for a client in the Kerberos database (or if the key registered

supports multiple encryption types; e.g. DES3-CBC-SHA1 and DES3-CBC-SHA1-KD), then the etype field from the AS request is used by the KDC to select the encryption method to be used to protect the encrypted part of the KRB_AS_REP message which is sent to the client. If there is more than one supported strong encryption type in the etype list, the first valid etype for which an encryption key is available is used. The encryption method used to protect the encrypted part of the KRB_TGS_REP message is the keytype of the session key found in the ticket granting ticket presented in the KRB_TGS_REQ.

If the user's key was generated using an alternate string to key function than that used by the selected encryption type, information needed by the string to key function will be returned to the client in the padata field of the KRB_AS_REP message using the PA-PW-SALT, PA-AFS3-SALT, or similar pre-authentication typed values. This does not affect the encryption performed by the KDC since the key stored in the principal database already has the string to key transformation applied.

When the etype field is present in a KDC request, whether an AS or TGS request, the KDC will attempt to assign the type of the random session key from the list of methods in the etype field. The KDC will select the appropriate type using the list of methods provided together with information from the Kerberos database indicating acceptable encryption methods for the application server. The KDC will not issue tickets with a weak session key encryption type.

If the requested start time is absent, indicates a time in the past, or is within the window of acceptable clock skew for the KDC and the POSTDATE option has not been specified, then the start time of the ticket is set to the authentication server's current time. If it indicates a time in the future beyond the acceptable clock skew, but the POSTDATED option has not been specified then the error KDC_ERR_CANNOT_POSTDATE is returned. Otherwise the requested start time is checked against the policy of the local realm (the administrator might decide to prohibit certain types or ranges of postdated tickets), and if acceptable, the ticket's start time is set as requested and the INVALID flag is set in the new ticket. The postdated ticket must be validated before use by presenting it to the KDC after the start time has been reached.

The expiration time of the ticket will be set to the earlier of the requested endtime and a time determined by local policy, possibly determined using realm or principal specific factors. For example, the expiration time may be set to the minimum of the following:

* The expiration time (endtime) requested in the KRB_AS_REQ message.
* The ticket's start time plus the maximum allowable lifetime associated with the client principal from the authentication server's database (see section 4).
* The ticket's start time plus the maximum allowable lifetime associated with the server principal.
* The ticket's start time plus the maximum lifetime set by the policy of the local realm.

If the requested expiration time minus the start time (as determined above) is less than a site-determined minimum lifetime, an error message with code KDC_ERR_NEVER_VALID is returned. If the requested expiration time for the ticket exceeds what was determined as above, and if the 'RENEWABLE-OK' option was requested, then the 'RENEWABLE' flag is set in the new ticket, and the renew-till value is set as if the 'RENEWABLE' option were requested (the field and option names are described fully in section 5.4.1).

If the RENEWABLE option has been requested or if the RENEWABLE-OK option has been set and a renewable ticket is to be issued, then the renew-till field

is set to the minimum of:

* Its requested value.
* The start time of the ticket plus the minimum of the two maximum renewable lifetimes associated with the principals' database entries.
* The start time of the ticket plus the maximum renewable lifetime set by the policy of the local realm.

The flags field of the new ticket will have the following options set if they have been requested and if the policy of the local realm allows: FORWARDABLE, MAY-POSTDATE, POSTDATED, PROXIABLE, RENEWABLE, ANONYMOUS. If the new ticket is post-dated (the start time is in the future), its INVALID flag will also be set.

If all of the above succeed, the server will encrypt ciphertext part of the ticket using the encryption key extracted from the server principal's record in the Kerberos database using the encryption type associated with the server principal's key (this choice is NOT affected by the etype field in the request). It then formats a KRB_AS_REP message (see section 5.4.2), copying the addresses in the request into the caddr of the response, placing any required pre-authentication data into the padata of the response, and encrypts the ciphertext part in the client's key using an acceptable encryption method requested in the etype field of the request, and sends the message to the client. See section A.2 for pseudocode.

3.1.4. Generation of KRB_ERROR message

Several errors can occur, and the Authentication Server responds by returning an error message, KRB_ERROR, to the client, with the error-code, e-text, and optional e-cksum fields set to appropriate values. The error message contents and details are described in Section 5.9.1.

3.1.5. Receipt of KRB_AS_REP message

If the reply message type is KRB_AS_REP, then the client verifies that the cname and crealm fields in the cleartext portion of the reply match what it requested. If any padata fields are present, they may be used to derive the proper secret key to decrypt the message. The client decrypts the encrypted part of the response using its secret key, verifies that the nonce in the encrypted part matches the nonce it supplied in its request (to detect replays). It also verifies that the sname and srealm in the response match those in the request (or are otherwise expected values), and that the host address field is also correct. It then stores the ticket, session key, start and expiration times, and other information for later use. The key-expiration field from the encrypted part of the response may be checked to notify the user of impending key expiration (the client program could then suggest remedial action, such as a password change). See section A.3 for pseudocode.

Proper decryption of the KRB_AS_REP message is not sufficient for the host to verify the identity of the user; the user and an attacker could cooperate to generate a KRB_AS_REP format message which decrypts properly but is not from the proper KDC. If the host wishes to verify the identity of the user, it must require the user to present application credentials which can be verified using a securely-stored secret key for the host. If those credentials can be verified, then the identity of the user can be assured.

3.1.6. Receipt of KRB_ERROR message

If the reply message type is KRB_ERROR, then the client interprets it as an error and performs whatever application-specific tasks are necessary to recover. If the client set the CANONICALIZE option and a KDC_ERR_WRONG_REALM

error was returned, the AS request should be retried to the realm and client principal name specified in the error message crealm and cname field respectively.

3.2. The Client/Server Authentication Exchange

Summary

Message direction	Message type	Section
Client to Application server	KRB_AP_REQ	5.5.1
[optional] Application server to client	KRB_AP_REP or KRB_ERROR	5.5.2 5.9.1

The client/server authentication (CS) exchange is used by network applications to authenticate the client to the server and vice versa. The client must have already acquired credentials for the server using the AS or TGS exchange.

3.2.1. The KRB_AP_REQ message

The KRB_AP_REQ contains authentication information which should be part of the first message in an authenticated transaction. It contains a ticket, an authenticator, and some additional bookkeeping information (see section 5.5.1 for the exact format). The ticket by itself is insufficient to authenticate a client, since tickets are passed across the network in cleartext[3.4], so the authenticator is used to prevent invalid replay of tickets by proving to the server that the client knows the session key of the ticket and thus is entitled to use the ticket. The KRB_AP_REQ message is referred to elsewhere as the 'authentication header.'

3.2.2. Generation of a KRB_AP_REQ message

When a client wishes to initiate authentication to a server, it obtains (either through a credentials cache, the AS exchange, or the TGS exchange) a ticket and session key for the desired service. The client may re-use any tickets it holds until they expire. To use a ticket the client constructs a new Authenticator from the system time, its name, and optionally an application specific checksum, an initial sequence number to be used in KRB_SAFE or KRB_PRIV messages, and/or a session subkey to be used in negotiations for a session key unique to this particular session. Authenticators may not be re-used and will be rejected if replayed to a server[3.5]. If a sequence number is to be included, it should be randomly chosen so that even after many messages have been exchanged it is not likely to collide with other sequence numbers in use.

The client may indicate a requirement of mutual authentication or the use of a session-key based ticket by setting the appropriate flag(s) in the ap-options field of the message.

The Authenticator is encrypted in the session key and combined with the ticket to form the KRB_AP_REQ message which is then sent to the end server along with any additional application-specific information. See section A.9 for pseudocode.

3.2.3. Receipt of KRB_AP_REQ message

Authentication is based on the server's current time of day (clocks must be loosely synchronized), the authenticator, and the ticket. Several errors are possible. If an error occurs, the server is expected to reply to the client with a KRB_ERROR message. This message may be encapsulated in the application protocol if its 'raw' form is not acceptable to the protocol. The format of error messages is described in section 5.9.1.

The algorithm for verifying authentication information is as follows. If the message type is not KRB_AP_REQ, the server returns the KRB_AP_ERR_MSG_TYPE error. If the key version indicated by the Ticket in the KRB_AP_REQ is not one the server can use (e.g., it indicates an old key, and the server no longer possesses a copy of the old key), the KRB_AP_ERR_BADKEYVER error is returned. If the USE-SESSION-KEY flag is set in the ap-options field, it indicates to the server that the ticket is encrypted in the session key from the server's ticket-granting ticket rather than its secret key [3.6].

Since it is possible for the server to be registered in multiple realms, with different keys in each, the srealm field in the unencrypted portion of the ticket in the KRB_AP_REQ is used to specify which secret key the server should use to decrypt that ticket. The KRB_AP_ERR_NOKEY error code is returned if the server doesn't have the proper key to decipher the ticket.

The ticket is decrypted using the version of the server's key specified by the ticket. If the decryption routines detect a modification of the ticket (each encryption system must provide safeguards to detect modified ciphertext; see section 6), the KRB_AP_ERR_BAD_INTEGRITY error is returned (chances are good that different keys were used to encrypt and decrypt).

The authenticator is decrypted using the session key extracted from the decrypted ticket. If decryption shows it to have been modified, the KRB_AP_ERR_BAD_INTEGRITY error is returned. The name and realm of the client from the ticket are compared against the same fields in the authenticator. If they don't match, the KRB_AP_ERR_BADMATCH error is returned (they might not match, for example, if the wrong session key was used to encrypt the authenticator). The addresses in the ticket (if any) are then searched for an address matching the operating-system reported address of the client. If no match is found or the server insists on ticket addresses but none are present in the ticket, the KRB_AP_ERR_BADADDR error is returned. If the local (server) time and the client time in the authenticator differ by more than the allowable clock skew (e.g., 5 minutes), the KRB_AP_ERR_SKEW error is returned.

Unless the application server provides its own suitable means to protect against replay (for example, a challenge-response sequence initiated by the server after authentication, or use of a server-generated encryption subkey), the server must utilize a replay cache to remember any authenticator presented within the allowable clock skew. Careful analysis of the application protocol and implementation is recommended before eliminating this cache. The replay cache will store the server name, along with the client name, time and microsecond fields from the recently-seen authenticators and if a matching tuple is found, the KRB_AP_ERR_REPEAT error is returned [3.7]. If a server loses track of authenticators presented within the allowable clock skew, it must reject all requests until the clock skew interval has passed, providing assurance that any lost or re-played authenticators will fall outside the allowable clock skew and can no longer be successfully replayed[3.8].

If a sequence number is provided in the authenticator, the server saves it for later use in processing KRB_SAFE and/or KRB_PRIV messages. If a subkey is present, the server either saves it for later use or uses it to help generate its own choice for a subkey to be returned in a KRB_AP_REP message.

If multiple servers (for example, different services on one machine, or a single service implemented on multiple machines) share a service principal (a practice we do not recommend in general, but acknowledge will be used in some cases), they should also share this replay cache, or the application protocol should be designed so as to eliminate the need for it. Note that this applies to all of the services, if any of the application protocols does not have replay protection built in; an authenticator used with such a

service could later be replayed to a different service with the same service principal but no replay protection, if the former doesn't record the authenticator information in the common replay cache.

The server computes the age of the ticket: local (server) time minus the start time inside the Ticket. If the start time is later than the current time by more than the allowable clock skew or if the INVALID flag is set in the ticket, the KRB_AP_ERR_TKT_NYV error is returned. Otherwise, if the current time is later than end time by more than the allowable clock skew, the KRB_AP_ERR_TKT_EXPIRED error is returned.

If all these checks succeed without an error, the server is assured that the client possesses the credentials of the principal named in the ticket and thus, the client has been authenticated to the server. See section A.10 for pseudocode.

Passing these checks provides only authentication of the named principal; it does not imply authorization to use the named service. Applications must make a separate authorization decisions based upon the authenticated name of the user, the requested operation, local access control information such as that contained in a .k5login or .k5users file, and possibly a separate distributed authorization service.

3.2.4. Generation of a KRB_AP_REP message

Typically, a client's request will include both the authentication information and its initial request in the same message, and the server need not explicitly reply to the KRB_AP_REQ. However, if mutual authentication (not only authenticating the client to the server, but also the server to the client) is being performed, the KRB_AP_REQ message will have MUTUAL-REQUIRED set in its ap-options field, and a KRB_AP_REP message is required in response. As with the error message, this message may be encapsulated in the application protocol if its "raw" form is not acceptable to the application's protocol. The timestamp and microsecond field used in the reply must be the client's timestamp and microsecond field (as provided in the authenticator)[3.9]. If a sequence number is to be included, it should be randomly chosen as described above for the authenticator. A subkey may be included if the server desires to negotiate a different subkey. The KRB_AP_REP message is encrypted in the session key extracted from the ticket. See section A.11 for pseudocode.

3.2.5. Receipt of KRB_AP_REP message

If a KRB_AP_REP message is returned, the client uses the session key from the credentials obtained for the server[3.10] to decrypt the message, and verifies that the timestamp and microsecond fields match those in the Authenticator it sent to the server. If they match, then the client is assured that the server is genuine. The sequence number and subkey (if present) are retained for later use. See section A.12 for pseudocode.

3.2.6. Using the encryption key

After the KRB_AP_REQ/KRB_AP_REP exchange has occurred, the client and server share an encryption key which can be used by the application. In some cases, the use of this session key will be implicit in the protocol; in others the method of use must be chosen from several alternatives. The 'true session key' to be used for KRB_PRIV, KRB_SAFE, or other application-specific uses may be chosen by the application based on the session key from the ticket and subkeys in the KRB_AP_REP message and the authenticator[3.11]. To mitigate the effect of failures in random number generation on the client it is strongly encouraged that any key derived by an application for subsequent

use include the full key entropy derived from the KDC generated session key carried in the ticket. We leave the protocol negotiations of how to use the key (e.g. selecting an encryption or checksum type) to the application programmer; the Kerberos protocol does not constrain the implementation options, but an example of how this might be done follows.

One way that an application may choose to negotiate a key to be used for subsequent integrity and privacy protection is for the client to propose a key in the subkey field of the authenticator. The server can then choose a key using the proposed key from the client as input, returning the new subkey in the subkey field of the application reply. This key could then be used for subsequent communication.

To make this example more concrete, if the communication patterns of an application dictates the use of encryption modes of operation incompatible with the encryption system used for the authenticator, then a key compatible with the required encryption system may be generated by either the client, the server, or collaboratively by both and exchanged using the subkey field. This generation might involve the use of a random number as a pre-key, initially generated by either party, which could then be encrypted using the session key from the ticket, and the result exchanged and used for subsequent encryption. By encrypting the pre-key with the session key from the ticket, randomness from the KDC generated key is assured of being present in the negotiated key. Application developers must be careful however, to use a means of introducing this entropy that does not allow an attacker to learn the session key from the ticket if it learns the key generated and used for subsequent communication. The reader should note that this is only an example, and that an analysis of the particular cryptosystem to be used, must be made before deciding how to generate values for the subkey fields, and the key to be used for subsequent communication.

With both the one-way and mutual authentication exchanges, the peers should take care not to send sensitive information to each other without proper assurances. In particular, applications that require privacy or integrity should use the KRB_AP_REP response from the server to client to assure both client and server of their peer's identity. If an application protocol requires privacy of its messages, it can use the KRB_PRIV message (section 3.5). The KRB_SAFE message (section 3.4) can be used to assure integrity.

3.3. The Ticket-Granting Service (TGS) Exchange

Summary

Message direction	Message type	Section
1. Client to Kerberos	KRB_TGS_REQ	5.4.1
2. Kerberos to client	KRB_TGS_REP or KRB_ERROR	5.4.2 5.9.1

The TGS exchange between a client and the Kerberos Ticket-Granting Server is initiated by a client when it wishes to obtain authentication credentials for a given server (which might be registered in a remote realm), when it wishes to renew or validate an existing ticket, or when it wishes to obtain a proxy ticket. In the first case, the client must already have acquired a ticket for the Ticket-Granting Service using the AS exchange (the ticket-granting ticket is usually obtained when a client initially authenticates to the system, such as when a user logs in). The message format for the TGS exchange is almost identical to that for the AS exchange. The primary difference is that encryption and decryption in the TGS exchange does not take place under the client's key. Instead, the session key from the ticket-granting ticket or renewable ticket, or sub-session key from an Authenticator is used. As is the case for all application servers, expired tickets are not accepted by the TGS, so once a renewable or ticket-granting ticket expires, the client must use a separate exchange to obtain valid tickets.

The TGS exchange consists of two messages: A request (KRB_TGS_REQ) from the client to the Kerberos Ticket-Granting Server, and a reply (KRB_TGS_REP or KRB_ERROR). The KRB_TGS_REQ message includes information authenticating the client plus a request for credentials. The authentication information consists of the authentication header (KRB_AP_REQ) which includes the client's previously obtained ticket-granting, renewable, or invalid ticket. In the ticket-granting ticket and proxy cases, the request may include one or more of: a list of network addresses, a collection of typed authorization data to be sealed in the ticket for authorization use by the application server, or additional tickets (the use of which are described later). The

TGS reply (KRB_TGS_REP) contains the requested credentials, encrypted in the session key from the ticket-granting ticket or renewable ticket, or if present, in the sub-session key from the Authenticator (part of the authentication header). The KRB_ERROR message contains an error code and text explaining what went wrong. The KRB_ERROR message is not encrypted. The KRB_TGS_REP message contains information which can be used to detect replays, and to associate it with the message to which it replies. The KRB_ERROR message also contains information which can be used to associate it with the message to which it replies, but except when an optional checksum is included in the KRB_ERROR message, it is not possible to detect replays or fabrications of such messages.

3.3.1. Generation of KRB_TGS_REQ message

Before sending a request to the ticket-granting service, the client must determine in which realm the application server is believed to be registered[3.12]. If the client knows the service principal name and realm and it does not already possess a ticket-granting ticket for the appropriate realm, then one must be obtained. This is first attempted by requesting a ticket-granting ticket for the destination realm from a Kerberos server for which the client possesses a ticket-granting ticket (using the KRB_TGS_REQ message recursively). The Kerberos server may return a TGT for the desired realm in which case one can proceed. Alternatively, the Kerberos server may return a TGT for a realm which is 'closer' to the desired realm (further along the standard hierarchical path between the client's realm and the requested realm server's realm).

If the client does not know the realm of the service or the true service principal name, then the CANONICALIZE option must be used in the request. This will cause the TGS to locate the service principal based on the target service name in the ticket and return the service principal name in the response. This function allows the KDC to inform the user of the registered Kerberos principal name and registered KDC for a server that may have more than one host name or whose registered realm can not be determined from the name of the host, but it is not to be used to locate the application server.

If the server name determined by a TGS supporting name canonicalization is with a remote KDC, then the response will include the principal name determined by the KDC, and will include a TGT for the remote realm or a realm 'closer' to the realm with which the server principal is registered. In this case, the canonicalization request must be repeated with a Kerberos server in the realm specified in the returned TGT. If neither are returned, then the request may be retried with a Kerberos server for a realm higher in the hierarchy. This request will itself require a ticket-granting ticket for the higher realm which must be obtained by recursively applying these directions.

Once the client obtains a ticket-granting ticket for the appropriate realm, it determines which Kerberos servers serve that realm, and contacts one. The list might be obtained through a configuration file or network service or it

may be generated from the name of the realm; as long as the secret keys exchanged by realms are kept secret, only denial of service results from using a false Kerberos server.

As in the AS exchange, the client may specify a number of options in the KRB_TGS_REQ message. The client prepares the KRB_TGS_REQ message, providing an authentication header as an element of the padata field, and including

the same fields as used in the KRB_AS_REQ message along with several optional fields: the enc-authorization-data field for application server use and additional tickets required by some options.

In preparing the authentication header, the client can select a sub-session key under which the response from the Kerberos server will be encrypted[3.13]. If the sub-session key is not specified, the session key from the ticket-granting ticket will be used. If the enc-authorization-data is present, it must be encrypted in the sub-session key, if present, from the authenticator portion of the authentication header, or if not present, using the session key from the ticket-granting ticket.

Once prepared, the message is sent to a Kerberos server for the destination realm. See section A.5 for pseudocode.

3.3.2. Receipt of KRB_TGS_REQ message

The KRB_TGS_REQ message is processed in a manner similar to the KRB_AS_REQ message, but there are many additional checks to be performed. First, the Kerberos server must determine which server the accompanying ticket is for and it must select the appropriate key to decrypt it. For a normal KRB_TGS_REQ message, it will be for the ticket granting service, and the TGS's key will be used. If the TGT was issued by another realm, then the appropriate inter-realm key must be used. If the accompanying ticket is not a ticket granting ticket for the current realm, but is for an application server in the current realm, the RENEW, VALIDATE, or PROXY options are specified in the request, and the server for which a ticket is requested is the server named in the accompanying ticket, then the KDC will decrypt the ticket in the authentication header using the key of the server for which it was issued. If no ticket can be found in the padata field, the KDC_ERR_PADATA_TYPE_NOSUPP error is returned.

Once the accompanying ticket has been decrypted, the user-supplied checksum in the Authenticator must be verified against the contents of the request, and the message rejected if the checksums do not match (with an error code of KRB_AP_ERR_MODIFIED) or if the checksum is not keyed or not collision-proof (with an error code of KRB_AP_ERR_INAPP_CKSUM). If the checksum type is not supported, the KDC_ERR_SUMTYPE_NOSUPP error is returned. If the authorization-data are present, they are decrypted using the sub-session key from the Authenticator.

If any of the decryptions indicate failed integrity checks, the KRB_AP_ERR_BAD_INTEGRITY error is returned. If the CANONICALIZE option is set in the KRB_TGS_REQ, then the requested service name might not be the true principal name or the service might not be in the TGS realm and the correct name must be determined.

3.3.3. Generation of KRB_TGS_REP message

The KRB_TGS_REP message shares its format with the KRB_AS_REP (KRB_KDC_REP), but with its type field set to KRB_TGS_REP. The detailed specification is in section 5.4.2.

The response will include a ticket for the requested server or for a ticket

granting server of an intermediate KDC to be contacted to obtain the requested ticket. The Kerberos database is queried to retrieve the record for the appropriate server (including the key with which the ticket will be encrypted). If the request is for a ticket granting ticket for a remote realm, and if no key is shared with the requested realm, then the Kerberos server will select the realm 'closest' to the requested realm with which it does share a key, and use that realm instead. If the CANONICALIZE option is set, the TGS may return a ticket containing the server name of the true service principal. If the requested server cannot be found in the TGS database, then a TGT for another trusted realm may be returned instead of a ticket for the service. This TGT is a referral mechanism to cause the client to retry the request to the realm of the TGT. These are the only cases where the response for the KDC will be for a different server than that requested by the client.

By default, the address field, the client's name and realm, the list of transited realms, the time of initial authentication, the expiration time, and the authorization data of the newly-issued ticket will be copied from the ticket-granting ticket (TGT) or renewable ticket. If the transited field needs to be updated, but the transited type is not supported, the KDC_ERR_TRTYPE_NOSUPP error is returned.

If the request specifies an endtime, then the endtime of the new ticket is set to the minimum of (a) that request, (b) the endtime from the TGT, and (c) the starttime of the TGT plus the minimum of the maximum life for the application server and the maximum life for the local realm (the maximum life for the requesting principal was already applied when the TGT was issued). If the new ticket is to be a renewal, then the endtime above is replaced by the minimum of (a) the value of the renew_till field of the ticket and (b) the starttime for the new ticket plus the life (endtime-starttime) of the old ticket.

If the FORWARDED option has been requested, then the resulting ticket will contain the addresses specified by the client. This option will only be honored if the FORWARDABLE flag is set in the TGT. The PROXY option is similar; the resulting ticket will contain the addresses specified by the client. It will be honored only if the PROXIABLE flag in the TGT is set. The PROXY option will not be honored on requests for additional ticket-granting tickets.

If the requested start time is absent, indicates a time in the past, or is within the window of acceptable clock skew for the KDC and the POSTDATE option has not been specified, then the start time of the ticket is set to the authentication server's current time. If it indicates a time in the future beyond the acceptable clock skew, but the POSTDATED option has not been specified or the MAY-POSTDATE flag is not set in the TGT, then the error KDC_ERR_CANNOT_POSTDATE is returned. Otherwise, if the ticket-granting ticket has the MAY-POSTDATE flag set, then the resulting ticket will be postdated and the requested starttime is checked against the policy of the local realm. If acceptable, the ticket's start time is set as requested, and the INVALID flag is set. The postdated ticket must be validated before use by presenting it to the KDC after the starttime has been reached. However, in no case may the starttime, endtime, or renew-till time of a newly-issued postdated ticket extend beyond the renew-till time of the ticket-granting ticket.

If the ENC-TKT-IN-SKEY option has been specified and an additional ticket has been included in the request, the KDC will decrypt the additional ticket using the key for the server to which the additional ticket was issued and verify that it is a ticket-granting ticket. If the name of the requested server is missing from the request, the name of the client in the additional ticket will be used. Otherwise the name of the requested server will be compared to the name of the client in the additional ticket and if

different, the request will be rejected. If the request succeeds, the session key from the additional ticket will be used to encrypt the new ticket that is issued instead of using the key of the server for which the new ticket will be used.

If the name of the server in the ticket that is presented to the KDC as part of the authentication header is not that of the ticket-granting server itself, the server is registered in the realm of the KDC, and the RENEW option is requested, then the KDC will verify that the RENEWABLE flag is set in the ticket, that the INVALID flag is not set in the ticket, and that the renew_till time is still in the future. If the VALIDATE option is requested, the KDC will check that the starttime has passed and the INVALID flag is set. If the PROXY option is requested, then the KDC will check that the PROXIABLE flag is set in the ticket. If the tests succeed, and the ticket passes the hotlist check described in the next section, the KDC will issue the appropriate new ticket.

The ciphertext part of the response in the KRB_TGS_REP message is encrypted in the sub-session key from the Authenticator, if present, or the session key from the ticket-granting ticket. It is not encrypted using the client's secret key. Furthermore, the client's key's expiration date and the key version number fields are left out since these values are stored along with the client's database record, and that record is not needed to satisfy a request based on a ticket-granting ticket. See section A.6 for pseudocode.

3.3.3.1. Checking for revoked tickets

Whenever a request is made to the ticket-granting server, the presented ticket(s) is(are) checked against a hot-list of tickets which have been canceled. This hot-list might be implemented by storing a range of issue timestamps for 'suspect tickets'; if a presented ticket had an authtime in that range, it would be rejected. In this way, a stolen ticket-granting ticket or renewable ticket cannot be used to gain additional tickets (renewals or otherwise) once the theft has been reported to the KDC for the realm in which the server resides. Any normal ticket obtained before it was reported stolen will still be valid (because they require no interaction with the KDC), but only until their normal expiration time. If TGT's have been issued for cross-realm authentication, use of the cross-realm TGT will not be affected unless the hot-list is propagated to the KDC's for the realms for which such cross-realm tickets were issued.

3.3.3.2. Encoding the transited field

If the identity of the server in the TGT that is presented to the KDC as part of the authentication header is that of the ticket-granting service, but the TGT was issued from another realm, the KDC will look up the inter-realm key shared with that realm and use that key to decrypt the ticket. If the ticket is valid, then the KDC will honor the request, subject to the constraints outlined above in the section describing the AS exchange. The realm part of the client's identity will be taken from the ticket-granting ticket. The name of the realm that issued the ticket-granting ticket will be added to the transited field of the ticket to be issued. This is accomplished by reading the transited field from the ticket-granting ticket (which is treated as an unordered set of realm names), adding the new realm to the set, then constructing and writing out its encoded (shorthand) form (this may involve a rearrangement of the existing encoding).

Note that the ticket-granting service does not add the name of its own realm. Instead, its responsibility is to add the name of the previous realm. This prevents a malicious Kerberos server from intentionally leaving out its own name (it could, however, omit other realms' names).

The names of neither the local realm nor the principal's realm are to be included in the transited field. They appear elsewhere in the ticket and both are known to have taken part in authenticating the principal. Since the endpoints are not included, both local and single-hop inter-realm authentication result in a transited field that is empty.

Because the name of each realm transited is added to this field, it might potentially be very long. To decrease the length of this field, its contents are encoded. The initially supported encoding is optimized for the normal case of inter-realm communication: a hierarchical arrangement of realms using either domain or X.500 style realm names. This encoding (called DOMAIN-X500-COMPRESS) is now described.

Realm names in the transited field are separated by a ",". The ",", "\", trailing "."s, and leading spaces (" ") are special characters, and if they are part of a realm name, they must be quoted in the transited field by preceding them with a "\".

A realm name ending with a "." is interpreted as being prepended to the previous realm. For example, we can encode traversal of EDU, MIT.EDU, ATHENA.MIT.EDU, WASHINGTON.EDU, and CS.WASHINGTON.EDU as:

"EDU,MIT.,ATHENA.,WASHINGTON.EDU,CS.".

Note that if ATHENA.MIT.EDU, or CS.WASHINGTON.EDU were end-points, that they would not be included in this field, and we would have:

"EDU,MIT.,WASHINGTON.EDU"

A realm name beginning with a "/" is interpreted as being appended to the previous realm[18]. If it is to stand by itself, then it should be preceded by a space (" "). For example, we can encode traversal of /COM/HP/APOLLO, /COM/HP, /COM, and /COM/DEC as:

"/COM,/HP,/APOLLO, /COM/DEC".

Like the example above, if /COM/HP/APOLLO and /COM/DEC are endpoints, they they would not be included in this field, and we would have:

"/COM,/HP"

A null subfield preceding or following a "," indicates that all realms between the previous realm and the next realm have been traversed[19]. Thus, "," means that all realms along the path between the client and the server have been traversed. ",EDU, /COM," means that that all realms from the client's realm up to EDU (in a domain style hierarchy) have been traversed, and that everything from /COM down to the server's realm in an X.500 style has also been traversed. This could occur if the EDU realm in one hierarchy shares an inter-realm key directly with the /COM realm in another hierarchy.

3.3.4. Receipt of KRB_TGS_REP message

When the KRB_TGS_REP is received by the client, it is processed in the same manner as the KRB_AS_REP processing described above. The primary difference is that the ciphertext part of the response must be decrypted using the session key from the ticket-granting ticket rather than the client's secret key. The server name returned in the reply is the true principal name of the service. See section A.7 for pseudocode.

3.4. The KRB_SAFE Exchange

The KRB_SAFE message may be used by clients requiring the ability to detect

modifications of messages they exchange. It achieves this by including a keyed collision-proof checksum of the user data and some control information. The checksum is keyed with an encryption key (usually the last key negotiated via subkeys, or the session key if no negotiation has occurred).

3.4.1. Generation of a KRB_SAFE message

When an application wishes to send a KRB_SAFE message, it collects its data and the appropriate control information and computes a checksum over them. The checksum algorithm should be a keyed one-way hash function (such as the RSA- MD5-DES checksum algorithm specified in section 6.4.5, or the DES MAC), generated using the sub-session key if present, or the session key. Different algorithms may be selected by changing the checksum type in the message. Unkeyed or non-collision-proof checksums are not suitable for this use.

The control information for the KRB_SAFE message includes both a timestamp and a sequence number. The designer of an application using the KRB_SAFE message must choose at least one of the two mechanisms. This choice should be based on the needs of the application protocol.

Sequence numbers are useful when all messages sent will be received by one's peer. Connection state is presently required to maintain the session key, so maintaining the next sequence number should not present an additional problem.

If the application protocol is expected to tolerate lost messages without them being resent, the use of the timestamp is the appropriate replay detection mechanism. Using timestamps is also the appropriate mechanism for multi-cast protocols where all of one's peers share a common sub-session key, but some messages will be sent to a subset of one's peers.

After computing the checksum, the client then transmits the information and checksum to the recipient in the message format specified in section 5.6.1.

3.4.2. Receipt of KRB_SAFE message

When an application receives a KRB_SAFE message, it verifies it as follows. If any error occurs, an error code is reported for use by the application.

The message is first checked by verifying that the protocol version and type fields match the current version and KRB_SAFE, respectively. A mismatch generates a KRB_AP_ERR_BADVERSION or KRB_AP_ERR_MSG_TYPE error. The application verifies that the checksum used is a collision-proof keyed checksum, and if it is not, a KRB_AP_ERR_INAPP_CKSUM error is generated. If the sender's address was included in the control information, the recipient verifies that the operating system's report of the sender's address matches the sender's address in the message, and (if a recipient address is specified or the recipient requires an address) that one of the recipient's addresses appears as the recipient's address in the message. A failed match for either case generates a KRB_AP_ERR_BADADDR error. Then the timestamp and usec and/or the sequence number fields are checked. If timestamp and usec are expected and not present, or they are present but not current, the KRB_AP_ERR_SKEW error is generated. If the server name, along with the client name, time and microsecond fields from the Authenticator match any recently-seen (sent or received[20]) such tuples, the KRB_AP_ERR_REPEAT error is generated. If an incorrect sequence number is included, or a sequence number is expected but not present, the KRB_AP_ERR_BADORDER error is generated. If neither a time-stamp and usec or a sequence number is present, a KRB_AP_ERR_MODIFIED error is generated. Finally, the checksum is

computed over the data and control information, and if it doesn't match the received checksum, a KRB_AP_ERR_MODIFIED error is generated.

If all the checks succeed, the application is assured that the message was generated by its peer and was not modified in transit.

3.5. The KRB_PRIV Exchange

The KRB_PRIV message may be used by clients requiring confidentiality and the ability to detect modifications of exchanged messages. It achieves this by encrypting the messages and adding control information.

3.5.1. Generation of a KRB_PRIV message

When an application wishes to send a KRB_PRIV message, it collects its data and the appropriate control information (specified in section 5.7.1) and encrypts them under an encryption key (usually the last key negotiated via subkeys, or the session key if no negotiation has occurred). As part of the control information, the client must choose to use either a timestamp or a sequence number (or both); see the discussion in section 3.4.1 for guidelines on which to use. After the user data and control information are encrypted, the client transmits the ciphertext and some 'envelope' information to the recipient.

3.5.2. Receipt of KRB_PRIV message

When an application receives a KRB_PRIV message, it verifies it as follows. If any error occurs, an error code is reported for use by the application.

The message is first checked by verifying that the protocol version and type fields match the current version and KRB_PRIV, respectively. A mismatch generates a KRB_AP_ERR_BADVERSION or KRB_AP_ERR_MSG_TYPE error. The

application then decrypts the ciphertext and processes the resultant plaintext. If decryption shows the data to have been modified, a KRB_AP_ERR_BAD_INTEGRITY error is generated. If the sender's address was included in the control information, the recipient verifies that the operating system's report of the sender's address matches the sender's address in the message, and (if a recipient address is specified or the recipient requires an address) that one of the recipient's addresses appears as the recipient's address in the message. A failed match for either case generates a KRB_AP_ERR_BADADDR error. Then the timestamp and usec and/or the sequence number fields are checked. If timestamp and usec are expected and not present, or they are present but not current, the KRB_AP_ERR_SKEW error is generated. If the server name, along with the client name, time and microsecond fields from the Authenticator match any recently-seen such tuples, the KRB_AP_ERR_REPEAT error is generated. If an incorrect sequence number is included, or a sequence number is expected but not present, the KRB_AP_ERR_BADORDER error is generated. If neither a time-stamp and usec or a sequence number is present, a KRB_AP_ERR_MODIFIED error is generated.

If all the checks succeed, the application can assume the message was generated by its peer, and was securely transmitted (without intruders able to see the unencrypted contents).

3.6. The KRB_CRED Exchange

The KRB_CRED message may be used by clients requiring the ability to send Kerberos credentials from one host to another. It achieves this by sending the tickets together with encrypted data containing the session keys and other information associated with the tickets.

3.6.1. Generation of a KRB_CRED message

When an application wishes to send a KRB_CRED message it first (using the KRB_TGS exchange) obtains credentials to be sent to the remote host. It then constructs a KRB_CRED message using the ticket or tickets so obtained, placing the session key needed to use each ticket in the key field of the corresponding KrbCredInfo sequence of the encrypted part of the KRB_CRED message.

Other information associated with each ticket and obtained during the KRB_TGS exchange is also placed in the corresponding KrbCredInfo sequence in the encrypted part of the KRB_CRED message. The current time and, if specifically required by the application the nonce, s-address, and r-address fields, are placed in the encrypted part of the KRB_CRED message which is then encrypted under an encryption key previously exchanged in the KRB_AP exchange (usually the last key negotiated via subkeys, or the session key if no negotiation has occurred).

3.6.2. Receipt of KRB_CRED message

When an application receives a KRB_CRED message, it verifies it. If any error occurs, an error code is reported for use by the application. The message is verified by checking that the protocol version and type fields match the current version and KRB_CRED, respectively. A mismatch generates a KRB_AP_ERR_BADVERSION or KRB_AP_ERR_MSG_TYPE error. The application then decrypts the ciphertext and processes the resultant plaintext. If decryption shows the data to have been modified, a KRB_AP_ERR_BAD_INTEGRITY error is generated.

If present or required, the recipient verifies that the operating system's report of the sender's address matches the sender's address in the message, and that one of the recipient's addresses appears as the recipient's address in the message. A failed match for either case generates a KRB_AP_ERR_BADADDR error. The timestamp and usec fields (and the nonce field if required) are checked next. If the timestamp and usec are not present, or they are present but not current, the KRB_AP_ERR_SKEW error is generated.

If all the checks succeed, the application stores each of the new tickets in its ticket cache together with the session key and other information in the corresponding KrbCredInfo sequence from the encrypted part of the KRB_CRED message.

4. The Kerberos Database

The Kerberos server must have access to a database containing the principal identifiers and secret keys of any principals to be authenticated[4.1] using such secret keys. The keying material in the database must be protected so that they are only accessible to the Kerberos server and administrative functions specifically authorized to access such material. Specific implementations may handle the storage of keying material separate from the Kerberos database (e.g. in hardware) or by encrypting the keying material before placing it in the Kerberos database. Some implementations might provide a means for using long term secret keys, but not for retrieving them from the Kerberos database.

4.1. Database contents

A database entry will typically contain the following fields, though in some instances a KDC may obtain these values through other means:

Field	Value
name	Principal's identifier
key	Principal's secret key
p_kvno	Principal's key version
max_life	Maximum lifetime for Tickets
max_renewable_life	Maximum total lifetime for renewable Tickets

The name field is an encoding of the principal's identifier. The key field contains an encryption key. This key is the principal's secret key. (The key can be encrypted before storage under a Kerberos "master key" to protect it in case the database is compromised but the master key is not. In that case, an extra field must be added to indicate the master key version used, see below.) The p_kvno field is the key version number of the principal's secret key. The max_life field contains the maximum allowable lifetime (endtime - starttime) for any Ticket issued for this principal. The max_renewable_life field contains the maximum allowable total lifetime for any renewable Ticket issued for this principal. (See section 3.1 for a description of how these lifetimes are used in determining the lifetime of a given Ticket.)

A server may provide KDC service to several realms, as long as the database representation provides a mechanism to distinguish between principal records with identifiers which differ only in the realm name.

When an application server's key changes, if the change is routine (i.e. not the result of disclosure of the old key), the old key should be retained by the server until all tickets that had been issued using that key have expired. Because of this, it is possible for several keys to be active for a single principal. Ciphertext encrypted in a principal's key is always tagged with the version of the key that was used for encryption, to help the recipient find the proper key for decryption.

When more than one key is active for a particular principal, the principal will have more than one record in the Kerberos database. The keys and key version numbers will differ between the records (the rest of the fields may or may not be the same). Whenever Kerberos issues a ticket, or responds to a request for initial authentication, the most recent key (known by the Kerberos server) will be used for encryption. This is the key with the highest key version number.

4.2. Additional fields

Project Athena's KDC implementation uses additional fields in its database:

Field	Value
K_kvno	Kerberos' key version
expiration	Expiration date for entry
attributes	Bit field of attributes
mod_date	Timestamp of last modification
mod_name	Modifying principal's identifier

The K_kvno field indicates the key version of the Kerberos master key under which the principal's secret key is encrypted.

After an entry's expiration date has passed, the KDC will return an error to any client attempting to gain tickets as or for the principal. (A database may want to maintain two expiration dates: one for the principal, and one for the principal's current key. This allows password aging to work independently of the principal's expiration date. However, due to the limited space in the responses, the KDC combines the key expiration and principal expiration date into a single value called 'key_exp', which is

used as a hint to the user to take administrative action.)

The attributes field is a bitfield used to govern the operations involving the principal. This field might be useful in conjunction with user registration procedures, for site-specific policy implementations (Project Athena currently uses it for their user registration process controlled by the system-wide database service, Moira [LGDSR87]), to identify whether a principal can play the role of a client or server or both, to note whether a server is appropriately trusted to receive credentials delegated by a client, or to identify the 'string to key' conversion algorithm used for a principal's key[4.2]. Other bits are used to indicate that certain ticket options should not be allowed in tickets encrypted under a principal's key (one bit each): Disallow issuing postdated tickets, disallow issuing forwardable tickets, disallow issuing tickets based on TGT authentication, disallow issuing renewable tickets, disallow issuing proxiable tickets, and disallow issuing tickets for which the principal is the server.

The mod_date field contains the time of last modification of the entry, and the mod_name field contains the name of the principal which last modified the entry.

4.3. Frequently Changing Fields

Some KDC implementations may wish to maintain the last time that a request was made by a particular principal. Information that might be maintained includes the time of the last request, the time of the last request for a ticket-granting ticket, the time of the last use of a ticket-granting ticket, or other times. This information can then be returned to the user in the last-req field (see section 5.2).

Other frequently changing information that can be maintained is the latest expiration time for any tickets that have been issued using each key. This field would be used to indicate how long old keys must remain valid to allow the continued use of outstanding tickets.

4.4. Site Constants

The KDC implementation should have the following configurable constants or options, to allow an administrator to make and enforce policy decisions:

* The minimum supported lifetime (used to determine whether the KDC_ERR_NEVER_VALID error should be returned). This constant should reflect reasonable expectations of round-trip time to the KDC, encryption/decryption time, and processing time by the client and target server, and it should allow for a minimum 'useful' lifetime.
* The maximum allowable total (renewable) lifetime of a ticket (renew_till - starttime).
* The maximum allowable lifetime of a ticket (endtime - starttime).
* Whether to allow the issue of tickets with empty address fields (including the ability to specify that such tickets may only be issued if the request specifies some authorization_data).
* Whether proxiable, forwardable, renewable or post-datable tickets are to be issued.

5. Message Specifications

This section (5) still has revisions that are pending based on comments by Tom Yu. Please see http://www.isi.edu/people/bcn/krb-revisions for the latest versions. There will be additional updates prior to the San Diego IETF meeting.

The following sections describe the exact contents and encoding of protocol

messages and objects. The ASN.1 base definitions are presented in the first subsection. The remaining subsections specify the protocol objects (tickets and authenticators) and messages. Specification of encryption and checksum techniques, and the fields related to them, appear in section 6.

Optional field in ASN.1 sequences

For optional integer value and date fields in ASN.1 sequences where a default value has been specified, certain default values will not be allowed in the encoding because these values will always be represented through defaulting by the absence of the optional field. For example, one will not send a microsecond zero value because one must make sure that there is only one way to encode this value.

Additional fields in ASN.1 sequences

Implementations receiving Kerberos messages with additional fields present in ASN.1 sequences should carry those fields through, unmodified, when the message is forwarded. Implementations should not drop such fields if the sequence is re-encoded.

5.1. ASN.1 Distinguished Encoding Representation

All uses of ASN.1 in Kerberos shall use the Distinguished Encoding Representation of the data elements as described in the X.509 specification, section 8.7 [X509-88].

5.2. ASN.1 Base Definitions

The following ASN.1 base definitions are used in the rest of this section. Note that since the underscore character (_) is not permitted in ASN.1 names, the hyphen (-) is used in its place for the purposes of ASN.1 names.

```
Realm ::=           GeneralString
PrincipalName ::=   SEQUENCE {
                    name-type[0]     INTEGER,
                    name-string[1]   SEQUENCE OF GeneralString
}
```

Kerberos realms are encoded as GeneralStrings. Realms shall not contain a character with the code 0 (the ASCII NUL). Most realms will usually consist of several components separated by periods (.), in the style of Internet Domain Names, or separated by slashes (/) in the style of X.500 names. Acceptable forms for realm names are specified in section 7. A PrincipalName is a typed sequence of components consisting of the following sub-fields:

name-type
: This field specifies the type of name that follows. Pre-defined values for this field are specified in section 7.2. The name-type should be treated as a hint. Ignoring the name type, no two names can be the same (i.e. at least one of the components, or the realm, must be different). This constraint may be eliminated in the future.

name-string
: This field encodes a sequence of components that form a name, each component encoded as a GeneralString. Taken together, a PrincipalName and a Realm form a principal identifier. Most PrincipalNames will have only a few components (typically one or two).

```
KerberosTime ::=   GeneralizedTime
                   -- Specifying UTC time zone (Z)
```

The timestamps used in Kerberos are encoded as GeneralizedTimes. An encoding shall specify the UTC time zone (Z) and shall not include any fractional portions of the seconds. It further shall not include any separators. Example: The only valid format for UTC time 6 minutes, 27 seconds after 9 pm on 6 November 1985 is 19851106210627Z.

```
HostAddress ::=     SEQUENCE  {
                    addr-type[0]            INTEGER,
                    address[1]              OCTET STRING
}

HostAddresses ::=   SEQUENCE OF HostAddress
```

The host address encodings consists of two fields:

addr-type
: This field specifies the type of address that follows. Pre-defined values for this field are specified in section 8.1.

address
: This field encodes a single address of type addr-type.

The two forms differ slightly. HostAddress contains exactly one address; HostAddresses contains a sequence of possibly many addresses.

```
AuthorizationData ::=   SEQUENCE OF SEQUENCE {
                        ad-type[0]              INTEGER,
                        ad-data[1]              OCTET STRING
}
```

ad-data
: This field contains authorization data to be interpreted according to the value of the corresponding ad-type field.

ad-type
: This field specifies the format for the ad-data subfield. All negative values are reserved for local use. Non-negative values are reserved for registered use.

Each sequence of type and data is referred to as an authorization element. Elements may be application specific, however, there is a common set of recursive elements that should be understood by all implementations. These elements contain other elements embedded within them, and the interpretation of the encapsulating element determines which of the embedded elements must be interpreted, and which may be ignored. Definitions for these common elements may be found in Appendix B.

```
TicketExtensions ::= SEQUENCE OF SEQUENCE {
           te-type[0]       INTEGER,
           te-data[1]       OCTET STRING
}
```

te-data
: This field contains opaque data that must be carried with the ticket to support extensions to the Kerberos protocol including but not limited to some forms of inter-realm key exchange and plaintext authorization data. See appendix C for some common uses of this field.

```
te-type
     This field specifies the format for the te-data subfield. All negative
     values are reserved for local use. Non-negative values are reserved for
     registered use.

APOptions ::=   BIT STRING
                  -- reserved(0),
                  -- use-session-key(1),
                  -- mutual-required(2)

TicketFlags ::= BIT STRING
                  -- reserved(0),
                  -- forwardable(1),
                  -- forwarded(2),
                  -- proxiable(3),
                  -- proxy(4),
                  -- may-postdate(5),
                  -- postdated(6),
                  -- invalid(7),
                  -- renewable(8),
                  -- initial(9),
                  -- pre-authent(10),
                  -- hw-authent(11),
                  -- transited-policy-checked(12),
                  -- ok-as-delegate(13)

KDCOptions ::=   BIT STRING io
                  -- reserved(0),
                  -- forwardable(1),
                  -- forwarded(2),
                  -- proxiable(3),
                  -- proxy(4),
                  -- allow-postdate(5),
                  -- postdated(6),
                  -- unused7(7),
                  -- renewable(8),
                  -- unused9(9),
                  -- unused10(10),
                  -- unused11(11),
                  -- unused12(12),
                  -- unused13(13),
                  -- requestanonymous(14),
                  -- canonicalize(15),
                  -- disable-transited-check(26),
                  -- renewable-ok(27),
                  -- enc-tkt-in-skey(28),
                  -- renew(30),
                  -- validate(31)

ASN.1 Bit strings have a length and a value. When used in Kerberos for the
APOptions, TicketFlags, and KDCOptions, the length of the bit string on
generated values should be the smallest number of bits needed to include the
highest order bit that is set (1), but in no case less than 32 bits. The
ASN.1 representation of the bit strings uses unnamed bits, with the meaning
of the individual bits defined by the comments in the specification above.
Implementations should accept values of bit strings of any length and treat
the value of flags corresponding to bits beyond the end of the bit string as
if the bit were reset (0). Comparison of bit strings of different length
should treat the smaller string as if it were padded with zeros beyond the
high order bits to the length of the longer string[23].
```

```
LastReq ::=   SEQUENCE OF SEQUENCE {
               lr-type[0]               INTEGER,
               lr-value[1]              KerberosTime
}
```

lr-type
This field indicates how the following lr-value field is to be interpreted. Negative values indicate that the information pertains only to the responding server. Non-negative values pertain to all servers for the realm. If the lr-type field is zero (0), then no information is conveyed by the lr-value subfield. If the absolute value of the lr-type field is one (1), then the lr-value subfield is the time of last initial request for a TGT. If it is two (2), then the lr-value subfield is the time of last initial request. If it is three (3), then the lr-value subfield is the time of issue for the newest ticket-granting ticket used. If it is four (4), then the lr-value subfield is the time of the last renewal. If it is five (5), then the lr-value subfield is the time of last request (of any type). If it is (6), then the lr-value subfield is the time when the password will expire.

lr-value
This field contains the time of the last request. the time must be interpreted according to the contents of the accompanying lr-type subfield.

See section 6 for the definitions of Checksum, ChecksumType, EncryptedData, EncryptionKey, EncryptionType, and KeyType.

5.3. Tickets and Authenticators

This section describes the format and encryption parameters for tickets and authenticators. When a ticket or authenticator is included in a protocol message it is treated as an opaque object.

5.3.1. Tickets

A ticket is a record that helps a client authenticate to a service. A Ticket contains the following information:

```
Ticket ::=        [APPLICATION 1] SEQUENCE {
                  tkt-vno[0]                  INTEGER,
                  realm[1]                    Realm,
                  sname[2]                    PrincipalName,
                  enc-part[3]                 EncryptedData, -- EncTicketPart
                  extensions[4]               TicketExtensions OPTIONAL
}

-- Encrypted part of ticket
EncTicketPart ::= [APPLICATION 3] SEQUENCE {
                  flags[0]                    TicketFlags,
                  key[1]                      EncryptionKey,
                  crealm[2]                   Realm,
                  cname[3]                    PrincipalName,
                  transited[4]                TransitedEncoding,
                  authtime[5]                 KerberosTime,
                  starttime[6]                KerberosTime OPTIONAL,
                  endtime[7]                  KerberosTime,
                  renew-till[8]               KerberosTime OPTIONAL,
                  caddr[9]                    HostAddresses OPTIONAL,
                  authorization-data[10]      AuthorizationData OPTIONAL
}
-- encoded Transited field
```

```
TransitedEncoding ::=   SEQUENCE {
                        tr-type[0]            INTEGER, -- must be registered
                        contents[1]           OCTET STRING
}
```

The encoding of EncTicketPart is encrypted in the key shared by Kerberos and the end server (the server's secret key). See section 6 for the format of the ciphertext.

tkt-vno
: This field specifies the version number for the ticket format. This document describes version number 5.

realm
: This field specifies the realm that issued a ticket. It also serves to identify the realm part of the server's principal identifier. Since a Kerberos server can only issue tickets for servers within its realm, the two will always be identical.

sname
: This field specifies all components of the name part of the server's identity, including those parts that identify a specific instance of a service.

enc-part
: This field holds the encrypted encoding of the EncTicketPart sequence.

extensions
: This optional field contains a sequence of extensions that may be used to carry information that must be carried with the ticket to support several extensions, including but not limited to plaintext authorization data, tokens for exchanging inter-realm keys, and other information that must be associated with a ticket for use by the application server. See Appendix C for definitions of common extensions.

  Note that some older versions of Kerberos did not support this field. Because this is an optional field it will not break older clients, but older clients might strip this field from the ticket before sending it to the application server. This limits the usefulness of this ticket field to environments where the ticket will not be parsed and reconstructed by these older Kerberos clients.

  If it is known that the client will strip this field from the ticket, as an interim measure the KDC may append this field to the end of the enc-part of the ticket and append a trailer indicating the length of the appended extensions field.

flags
: This field indicates which of various options were used or requested when the ticket was issued. It is a bit-field, where the selected options are indicated by the bit being set (1), and the unselected options and reserved fields being reset (0). Bit 0 is the most significant bit. The encoding of the bits is specified in section 5.2. The flags are described in more detail above in section 2. The meanings of the flags are:

Bits	Name	Description
0	RESERVED	Reserved for future expansion of this field.
1	FORWARDABLE	The FORWARDABLE flag is normally only interpreted by the TGS, and can be ignored by end servers. When set, this flag tells the ticket-granting server that it is OK to issue a new ticket-granting ticket with a different network address based on the

		presented ticket.
2	FORWARDED	When set, this flag indicates that the ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.
3	PROXIABLE	The PROXIABLE flag is normally only interpreted by the TGS, and can be ignored by end servers. The PROXIABLE flag has an interpretation identical to that of the FORWARDABLE flag, except that the PROXIABLE flag tells the ticket-granting server that only non-ticket-granting tickets may be issued with different network addresses.
4	PROXY	When set, this flag indicates that a ticket is a proxy.
5	MAY-POSTDATE	The MAY-POSTDATE flag is normally only interpreted by the TGS, and can be ignored by end servers. This flag tells the ticket-granting server that a post-dated ticket may be issued based on this ticket-granting ticket.
6	POSTDATED	This flag indicates that this ticket has been postdated. The end-service can check the authtime field to see when the original authentication occurred.
7	INVALID	This flag indicates that a ticket is invalid, and it must be validated by the KDC before use. Application servers must reject tickets which have this flag set.
8	RENEWABLE	The RENEWABLE flag is normally only interpreted by the TGS, and can usually be ignored by end servers (some particularly careful servers may wish to disallow renewable tickets). A renewable ticket can be used to obtain a replacement ticket that expires at a later date.
9	INITIAL	This flag indicates that this ticket was issued using the AS protocol, and not issued based on a ticket-granting ticket.
10	PRE-AUTHENT	This flag indicates that during initial authentication, the client was authenticated by the KDC before a ticket was issued. The strength of the preauthentication method is not indicated, but is acceptable to the

		KDC.
11	HW-AUTHENT	This flag indicates that the protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client. The hardware authentication method is selected by the KDC and the strength of the method is not indicated.
12	TRANSITED-POLICY-CHECKED	This flag indicates that the KDC for the realm has checked the transited field against a realm defined policy for trusted certifiers. If this flag is reset (0), then the application server must check the transited field itself, and if unable to do so it must reject the authentication. If the flag is set (1) then the application server may skip its own validation of the transited field, relying on the validation performed by the KDC. At its option the application server may still apply its own validation based on a separate policy for acceptance.
13	OK-AS-DELEGATE	This flag indicates that the server (not the client) specified in the ticket has been determined by policy of the realm to be a suitable recipient of delegation. A client can use the presence of this flag to help it make a decision whether to delegate credentials (either grant a proxy or a forwarded ticket granting ticket) to this server. The client is free to ignore the value of this flag. When setting this flag, an administrator should consider the Security and placement of the server on which the service will run, as well as whether the service requires the use of delegated credentials.
14	ANONYMOUS	This flag indicates that the principal named in the ticket is a generic principal for the realm and does not identify the individual using the ticket. The purpose of the ticket is only to securely distribute a session key, and not to identify the user. Subsequent requests using the same ticket and session may be considered as originating from the same user, but requests with the same username but a different ticket are likely to originate from different users.
15-31	RESERVED	Reserved for future use.

key
This field exists in the ticket and the KDC response and is used to pass the session key from Kerberos to the application server and the client. The field's encoding is described in section 6.2.

crealm
This field contains the name of the realm in which the client is registered and in which initial authentication took place.

cname
This field contains the name part of the client's principal identifier.

transited
This field lists the names of the Kerberos realms that took part in authenticating the user to whom this ticket was issued. It does not specify the order in which the realms were transited. See section 3.3.3.2 for details on how this field encodes the traversed realms. When the names of CA's are to be embedded in the transited field (as specified for some extensions to the protocol), the X.500 names of the CA's should be mapped into items in the transited field using the mapping defined by RFC2253.

authtime
This field indicates the time of initial authentication for the named principal. It is the time of issue for the original ticket on which this ticket is based. It is included in the ticket to provide additional information to the end service, and to provide the necessary information for implementation of a `hot list' service at the KDC. An end service that is particularly paranoid could refuse to accept tickets for which the initial authentication occurred "too far" in the past. This field is also returned as part of the response from the KDC. When returned as part of the response to initial authentication (KRB_AS_REP), this is the current time on the Kerberos server[24].

starttime
This field in the ticket specifies the time after which the ticket is valid. Together with endtime, this field specifies the life of the ticket. If it is absent from the ticket, its value should be treated as that of the authtime field.

endtime
This field contains the time after which the ticket will not be honored (its expiration time). Note that individual services may place their own limits on the life of a ticket and may reject tickets which have not yet expired. As such, this is really an upper bound on the expiration time for the ticket.

renew-till
This field is only present in tickets that have the RENEWABLE flag set in the flags field. It indicates the maximum endtime that may be included in a renewal. It can be thought of as the absolute expiration time for the ticket, including all renewals.

caddr
This field in a ticket contains zero (if omitted) or more (if present) host addresses. These are the addresses from which the ticket can be used. If there are no addresses, the ticket can be used from any location. The decision by the KDC to issue or by the end server to accept zero-address tickets is a policy decision and is left to the Kerberos and end-service administrators; they may refuse to issue or accept such tickets. The suggested and default policy, however, is that such tickets will only be issued or accepted when additional information that can be used to restrict the use of the ticket is included in the authorization_data field. Such a ticket is a capability.

Network addresses are included in the ticket to make it harder for an attacker to use stolen credentials. Because the session key is not sent over the network in cleartext, credentials can't be stolen simply by listening to the network; an attacker has to gain access to the session key (perhaps through operating system security breaches or a careless

user's unattended session) to make use of stolen tickets.

It is important to note that the network address from which a connection is received cannot be reliably determined. Even if it could be, an attacker who has compromised the client's workstation could use the credentials from there. Including the network addresses only makes it more difficult, not impossible, for an attacker to walk off with stolen credentials and then use them from a "safe" location.

authorization-data

The authorization-data field is used to pass authorization data from the principal on whose behalf a ticket was issued to the application service. If no authorization data is included, this field will be left out. Experience has shown that the name of this field is confusing, and that a better name for this field would be restrictions. Unfortunately, it is not possible to change the name of this field at this time.

This field contains restrictions on any authority obtained on the basis of authentication using the ticket. It is possible for any principal in possession of credentials to add entries to the authorization data field since these entries further restrict what can be done with the ticket. Such additions can be made by specifying the additional entries when a new ticket is obtained during the TGS exchange, or they may be added during chained delegation using the authorization data field of the authenticator.

Because entries may be added to this field by the holder of credentials, except when an entry is separately authenticated by encapsulation in the kdc-issued element, it is not allowable for the presence of an entry in the authorization data field of a ticket to amplify the privileges one would obtain from using a ticket.

The data in this field may be specific to the end service; the field will contain the names of service specific objects, and the rights to those objects. The format for this field is described in section 5.2. Although Kerberos is not concerned with the format of the contents of the sub-fields, it does carry type information (ad-type).

By using the authorization_data field, a principal is able to issue a proxy that is valid for a specific purpose. For example, a client wishing to print a file can obtain a file server proxy to be passed to the print server. By specifying the name of the file in the authorization_data field, the file server knows that the print server can only use the client's rights when accessing the particular file to be printed.

A separate service providing authorization or certifying group membership may be built using the authorization-data field. In this case, the entity granting authorization (not the authorized entity), may obtain a ticket in its own name (e.g. the ticket is issued in the name of a privilege server), and this entity adds restrictions on its own authority and delegates the restricted authority through a proxy to the client. The client would then present this authorization credential to the application server separately from the authentication exchange. Alternatively, such authorization credentials may be embedded in the ticket authenticating the authorized entity, when the authorization is separately authenticated using the kdc-issued authorization data element (see B.4).

Similarly, if one specifies the authorization-data field of a proxy and leaves the host addresses blank, the resulting ticket and session key can be treated as a capability. See [Neu93] for some suggested uses of this field.

The authorization-data field is optional and does not have to be included in a ticket.

5.3.2. Authenticators

An authenticator is a record sent with a ticket to a server to certify the client's knowledge of the encryption key in the ticket, to help the server detect replays, and to help choose a "true session key" to use with the particular session. The encoding is encrypted in the ticket's session key shared by the client and the server:

```
-- Unencrypted authenticator
Authenticator ::= [APPLICATION 2] SEQUENCE  {
                  authenticator-vno[0]          INTEGER,
                  crealm[1]                     Realm,
                  cname[2]                      PrincipalName,
                  cksum[3]                      Checksum OPTIONAL,
                  cusec[4]                      INTEGER,
                  ctime[5]                      KerberosTime,
                  subkey[6]                     EncryptionKey OPTIONAL,
                  seq-number[7]                 INTEGER OPTIONAL,
                  authorization-data[8]         AuthorizationData OPTIONAL
}
```

authenticator-vno
: This field specifies the version number for the format of the authenticator. This document specifies version 5.

crealm and cname
: These fields are the same as those described for the ticket in section 5.3.1.

cksum
: This field contains a checksum of the application data that accompanies the KRB_AP_REQ.

cusec
: This field contains the microsecond part of the client's timestamp. Its value (before encryption) ranges from 0 to 999999. It often appears along with ctime. The two fields are used together to specify a reasonably accurate timestamp.

ctime
: This field contains the current time on the client's host.

subkey
: This field contains the client's choice for an encryption key which is to be used to protect this specific application session. Unless an application specifies otherwise, if this field is left out the session key from the ticket will be used.

seq-number
: This optional field includes the initial sequence number to be used by the KRB_PRIV or KRB_SAFE messages when sequence numbers are used to detect replays (It may also be used by application specific messages). When included in the authenticator this field specifies the initial sequence number for messages from the client to the server. When included in the AP-REP message, the initial sequence number is that for messages from the server to the client. When used in KRB_PRIV or KRB_SAFE messages, it is incremented by one after each message is sent. Sequence numbers fall in the range of 0 through 2^32 - 1 and wrap to zero following the value 2^32 - 1.

  For sequence numbers to adequately support the detection of replays they should be non-repeating, even across connection boundaries. The initial sequence number should be random and uniformly distributed across the full space of possible sequence numbers, so that it cannot

be guessed by an attacker and so that it and the successive sequence numbers do not repeat other sequences.

authorization-data
This field is the same as described for the ticket in section 5.3.1. It is optional and will only appear when additional restrictions are to be placed on the use of a ticket, beyond those carried in the ticket itself.

5.4. Specifications for the AS and TGS exchanges

This section specifies the format of the messages used in the exchange between the client and the Kerberos server. The format of possible error messages appears in section 5.9.1.

5.4.1. KRB_KDC_REQ definition

The KRB_KDC_REQ message has no type of its own. Instead, its type is one of KRB_AS_REQ or KRB_TGS_REQ depending on whether the request is for an initial ticket or an additional ticket. In either case, the message is sent from the client to the Authentication Server to request credentials for a service.

The message fields are:

```
AS-REQ ::=          [APPLICATION 10] KDC-REQ
TGS-REQ ::=         [APPLICATION 12] KDC-REQ

KDC-REQ ::=         SEQUENCE {
                    pvno[1]             INTEGER,
                    msg-type[2]         INTEGER,
                    padata[3]           SEQUENCE OF PA-DATA OPTIONAL,
                    req-body[4]         KDC-REQ-BODY
}

PA-DATA ::=         SEQUENCE {
                    padata-type[1]      INTEGER,
                    padata-value[2]     OCTET STRING,
                                        -- might be encoded AP-REQ
}

KDC-REQ-BODY ::=    SEQUENCE {
                     kdc-options[0]         KDCOptions,
                     cname[1]               PrincipalName OPTIONAL,
                                            -- Used only in AS-REQ
                     realm[2]               Realm, -- Server's realm
                                            -- Also client's in AS-REQ
                     sname[3]               PrincipalName OPTIONAL,
                     from[4]                KerberosTime OPTIONAL,
                     till[5]                KerberosTime OPTIONAL,
                     rtime[6]               KerberosTime OPTIONAL,
                     nonce[7]               INTEGER,
                     etype[8]               SEQUENCE OF INTEGER,
                                            -- EncryptionType,
                                            -- in preference order
                     addresses[9]           HostAddresses OPTIONAL,
                 enc-authorization-data[10] EncryptedData OPTIONAL,
                                            -- Encrypted AuthorizationData
                                            -- encoding
                     additional-tickets[11] SEQUENCE OF Ticket OPTIONAL
}
```

The fields in this message are:

pvno

This field is included in each message, and specifies the protocol version number. This document specifies protocol version 5.

msg-type

This field indicates the type of a protocol message. It will almost always be the same as the application identifier associated with a message. It is included to make the identifier more readily accessible to the application. For the KDC-REQ message, this type will be KRB_AS_REQ or KRB_TGS_REQ.

padata

The padata (pre-authentication data) field contains a sequence of authentication information which may be needed before credentials can be issued or decrypted. In the case of requests for additional tickets (KRB_TGS_REQ), this field will include an element with padata-type of PA-TGS-REQ and data of an authentication header (ticket-granting ticket and authenticator). The checksum in the authenticator (which must be collision-proof) is to be computed over the KDC-REQ-BODY encoding. In most requests for initial authentication (KRB_AS_REQ) and most replies (KDC-REP), the padata field will be left out.

This field may also contain information needed by certain extensions to the Kerberos protocol. For example, it might be used to initially verify the identity of a client before any response is returned. When this field is used to authenticate or pre-authenticate a request, it should contain a keyed checksum over the KDC-REQ-BODY to bind the pre-authentication data to rest of the request. The KDC, as a matter of policy, may decide whether to honor a KDC-REQ which includes any pre-authentication data that does not contain the checksum field.

PA-ENC-TIMESTAMP defines a pre-authentication data type that is used for authenticating a client by way of an encrypted timestamp. This is accomplished with a padata field with padata-type equal to PA-ENC-TIMESTAMP and padata-value defined as follows (query: the checksum is new in this definition. If the optional field will break things we can keep the old PA-ENC-TS-ENC, and define a new alternate form that includes the checksum). :

```
padata-type     ::= PA-ENC-TIMESTAMP
padata-value    ::= EncryptedData -- PA-ENC-TS-ENC

PA-ENC-TS-ENC   ::= SEQUENCE {
                    patimestamp[0]     KerberosTime, -- client's time
                    pausec[1]          INTEGER OPTIONAL,
                    pachecksum[2]      checksum OPTIONAL
                                       -- keyed checksum of KDC-REQ-BODY
}
```

with patimestamp containing the client's time and pausec containing the microseconds which may be omitted if a client will not generate more than one request per second. The ciphertext (padata-value) consists of the PA-ENC-TS-ENC sequence, encrypted using the client's secret key.

It may also be used by the client to specify the version of a key that is being used for accompanying preauthentication, and/or which should be used to encrypt the reply from the KDC.

```
padata-type     ::= PA-USE-SPECIFIED-KVNO
padata-value    ::= Integer
}
```

The KDC should only accept and abide by the value of the use-specified-kvno preauthentication data field when the specified key is still valid and until use of a new key is confirmed. This situation

is likely to occur primarily during the period during which an updated key is propagating to other KDC's in a realm.

The padata field can also contain information needed to help the KDC or the client select the key needed for generating or decrypting the response. This form of the padata is useful for supporting the use of certain token cards with Kerberos. The details of such extensions are specified in separate documents. See [Pat92] for additional uses of this field.

padata-type
The padata-type element of the padata field indicates the way that the padata-value element is to be interpreted. Negative values of padata-type are reserved for unregistered use; non-negative values are used for a registered interpretation of the element type.

req-body
This field is a placeholder delimiting the extent of the remaining fields. If a checksum is to be calculated over the request, it is calculated over an encoding of the KDC-REQ-BODY sequence which is enclosed within the req-body field.

kdc-options
This field appears in the KRB_AS_REQ and KRB_TGS_REQ requests to the KDC and indicates the flags that the client wants set on the tickets as well as other information that is to modify the behavior of the KDC. Where appropriate, the name of an option may be the same as the flag that is set by that option. Although in most case, the bit in the options field will be the same as that in the flags field, this is not guaranteed, so it is not acceptable to simply copy the options field to the flags field. There are various checks that must be made before honoring an option anyway.

The kdc_options field is a bit-field, where the selected options are indicated by the bit being set (1), and the unselected options and reserved fields being reset (0). The encoding of the bits is specified in section 5.2. The options are described in more detail above in section 2. The meanings of the options are:

Bits	Name	Description
0	RESERVED	Reserved for future expansion of this field.
1	FORWARDABLE	The FORWARDABLE option indicates that the ticket to be issued is to have its forwardable flag set. It may only be set on the initial request, or in a subsequent request if the ticket-granting ticket on which it is based is also forwardable.
2	FORWARDED	The FORWARDED option is only specified in a request to the ticket-granting server and will only be honored if the ticket-granting ticket in the request has its FORWARDABLE bit set. This option indicates that this is a request for forwarding. The address(es) of the host from which the resulting ticket is to be valid are included in the addresses field of the request.

3	PROXIABLE	The PROXIABLE option indicates that the ticket to be issued is to have its proxiable flag set. It may only be set on the initial request, or in a subsequent request if the ticket-granting ticket on which it is based is also proxiable.
4	PROXY	The PROXY option indicates that this is a request for a proxy. This option will only be honored if the ticket-granting ticket in the request has its PROXIABLE bit set. The address(es) of the host from which the resulting ticket is to be valid are included in the addresses field of the request.
5	ALLOW-POSTDATE	The ALLOW-POSTDATE option indicates that the ticket to be issued is to have its MAY-POSTDATE flag set. It may only be set on the initial request, or in a subsequent request if the ticket-granting ticket on which it is based also has its MAY-POSTDATE flag set.
6	POSTDATED	The POSTDATED option indicates that this is a request for a postdated ticket. This option will only be honored if the ticket-granting ticket on which it is based has its MAY-POSTDATE flag set. The resulting ticket will also have its INVALID flag set, and that flag may be reset by a subsequent request to the KDC after the starttime in the ticket has been reached.
7	UNUSED	This option is presently unused.
8	RENEWABLE	The RENEWABLE option indicates that the ticket to be issued is to have its RENEWABLE flag set. It may only be set on the initial request, or when the ticket-granting ticket on which the request is based is also renewable. If this option is requested, then the rtime field in the request contains the desired absolute expiration time for the ticket.
9	RESERVED	Reserved for PK-Cross
10-13	UNUSED	These options are presently unused.
		The REQUEST-ANONYMOUS option indicates that the ticket to be issued is not to identify the user

		to which it was issued. Instead, the principal identifier is to be generic, as specified by the policy of the realm (e.g. usually
14	REQUEST-ANONYMOUS	anonymous@realm). The purpose of the ticket is only to securely distribute a session key, and not to identify the user. The ANONYMOUS flag on the ticket to be returned should be set. If the local realms policy does not permit anonymous credentials, the request is to be rejected.
15	CANONICALIZE	The CANONICALIZE option indicates that the client will accept the return of a true server name instead of the name specified in the request. In addition the client will be able to process any TGT referrals that will direct the client to another realm to locate the requested server. If a KDC does not support name- canonicalization, the option is ignored and the appropriate KDC_ERR_C_PRINCIPAL_UNKNOWN or KDC_ERR_S_PRINCIPAL_UNKNOWN error is returned. [JBrezak]
16-25	RESERVED	Reserved for future use.
		By default the KDC will check the transited field of a ticket-granting-ticket against the policy of the local realm before it will issue derivative tickets based on the ticket granting ticket. If this flag is set in the request, checking of the transited field is
26	DISABLE-TRANSITED-CHECK	disabled. Tickets issued without the performance of this check will be noted by the reset (0) value of the TRANSITED-POLICY-CHECKED flag, indicating to the application server that the transited field must be checked locally. KDC's are encouraged but not required to honor the DISABLE-TRANSITED-CHECK option.
27	RENEWABLE-OK	The RENEWABLE-OK option indicates that a renewable ticket will be acceptable if a ticket with the requested life cannot otherwise be provided. If a ticket with the requested life cannot be provided, then a renewable ticket may be issued with a renew-till equal to the requested endtime. The value of the renew-till field may still be limited by local limits, or limits

		selected by the individual principal or server.
28	ENC-TKT-IN-SKEY	This option is used only by the ticket-granting service. The ENC-TKT-IN-SKEY option indicates that the ticket for the end server is to be encrypted in the session key from the additional ticket-granting ticket provided.
29	RESERVED	Reserved for future use.
30	RENEW	This option is used only by the ticket-granting service. The RENEW option indicates that the present request is for a renewal. The ticket provided is encrypted in the secret key for the server on which it is valid. This option will only be honored if the ticket to be renewed has its RENEWABLE flag set and if the time in its renew-till field has not passed. The ticket to be renewed is passed in the padata field as part of the authentication header.
31	VALIDATE	This option is used only by the ticket-granting service. The VALIDATE option indicates that the request is to validate a postdated ticket. It will only be honored if the ticket presented is postdated, presently has its INVALID flag set, and would be otherwise usable at this time. A ticket cannot be validated before its starttime. The ticket presented for validation is encrypted in the key of the server for which it is valid and is passed in the padata field as part of the authentication header.

cname and sname

These fields are the same as those described for the ticket in section 5.3.1. sname may only be absent when the ENC-TKT-IN-SKEY option is specified. If absent, the name of the server is taken from the name of the client in the ticket passed as additional-tickets.

enc-authorization-data

The enc-authorization-data, if present (and it can only be present in the TGS_REQ form), is an encoding of the desired authorization-data encrypted under the sub-session key if present in the Authenticator, or alternatively from the session key in the ticket-granting ticket, both from the padata field in the KRB_AP_REQ.

realm

This field specifies the realm part of the server's principal identifier. In the AS exchange, this is also the realm part of the client's principal identifier. If the CANONICALIZE option is set, the realm is used as a hint to the KDC for its database lookup.

from

This field is included in the KRB_AS_REQ and KRB_TGS_REQ ticket requests when the requested ticket is to be postdated. It specifies the

desired start time for the requested ticket. If this field is omitted then the KDC should use the current time instead.

till
: This field contains the expiration date requested by the client in a ticket request. It is optional and if omitted the requested ticket is to have the maximum endtime permitted according to KDC policy for the parties to the authentication exchange as limited by expiration date of the ticket granting ticket or other preauthentication credentials.

rtime
: This field is the requested renew-till time sent from a client to the KDC in a ticket request. It is optional.

nonce
: This field is part of the KDC request and response. It intended to hold a random number generated by the client. If the same number is included in the encrypted response from the KDC, it provides evidence that the response is fresh and has not been replayed by an attacker. Nonces must never be re-used. Ideally, it should be generated randomly, but if the correct time is known, it may suffice[25].

etype
: This field specifies the desired encryption algorithm to be used in the response.

addresses
: This field is included in the initial request for tickets, and optionally included in requests for additional tickets from the ticket-granting server. It specifies the addresses from which the requested ticket is to be valid. Normally it includes the addresses for the client's host. If a proxy is requested, this field will contain other addresses. The contents of this field are usually copied by the KDC into the caddr field of the resulting ticket.

additional-tickets
: Additional tickets may be optionally included in a request to the ticket-granting server. If the ENC-TKT-IN-SKEY option has been specified, then the session key from the additional ticket will be used in place of the server's key to encrypt the new ticket. When he ENC-TKT-IN-SKEY option is used for user-to-user authentication, this additional ticket may be a TGT issued by the local realm or an inter-realm TGT issued for the current KDC's realm by a remote KDC. If more than one option which requires additional tickets has been specified, then the additional tickets are used in the order specified by the ordering of the options bits (see kdc-options, above).

The application code will be either ten (10) or twelve (12) depending on whether the request is for an initial ticket (AS-REQ) or for an additional ticket (TGS-REQ).

The optional fields (addresses, authorization-data and additional-tickets) are only included if necessary to perform the operation specified in the kdc-options field.

It should be noted that in KRB_TGS_REQ, the protocol version number appears twice and two different message types appear: the KRB_TGS_REQ message contains these fields as does the authentication header (KRB_AP_REQ) that is passed in the padata field.

5.4.2. KRB_KDC_REP definition

The KRB_KDC_REP message format is used for the reply from the KDC for either an initial (AS) request or a subsequent (TGS) request. There is no message type for KRB_KDC_REP. Instead, the type will be either KRB_AS_REP or KRB_TGS_REP. The key used to encrypt the ciphertext part of the reply depends on the message type. For KRB_AS_REP, the ciphertext is encrypted in

```
the client's secret key, and the client's key version number is included in
the key version number for the encrypted data. For KRB_TGS_REP, the
ciphertext is encrypted in the sub-session key from the Authenticator, or if
absent, the session key from the ticket-granting ticket used in the request.
In that case, no version number will be present in the EncryptedData
sequence.



The KRB_KDC_REP message contains the following fields:

AS-REP ::=    [APPLICATION 11] KDC-REP
TGS-REP ::=   [APPLICATION 13] KDC-REP

KDC-REP ::=   SEQUENCE {
              pvno[0]                   INTEGER,
              msg-type[1]               INTEGER,
              padata[2]                 SEQUENCE OF PA-DATA OPTIONAL,
              crealm[3]                 Realm,
              cname[4]                  PrincipalName,
              ticket[5]                 Ticket,
              enc-part[6]               EncryptedData
                                        -- EncASREpPart or EncTGSReoOart
}

EncASRepPart ::=    [APPLICATION 25[27]] EncKDCRepPart
EncTGSRepPart ::=   [APPLICATION 26] EncKDCRepPart

EncKDCRepPart ::=   SEQUENCE {
                    key[0]              EncryptionKey,
                    last-req[1]         LastReq,
                    nonce[2]            INTEGER,
                    key-expiration[3]   KerberosTime OPTIONAL,
                    flags[4]            TicketFlags,
                    authtime[5]         KerberosTime,
                    starttime[6]        KerberosTime OPTIONAL,
                    endtime[7]          KerberosTime,
                    renew-till[8]       KerberosTime OPTIONAL,
                    srealm[9]           Realm,
                    sname[10]           PrincipalName,
                    caddr[11]           HostAddresses OPTIONAL
}

pvno and msg-type
     These fields are described above in section 5.4.1. msg-type is either
     KRB_AS_REP or KRB_TGS_REP.
padata
     This field is described in detail in section 5.4.1. One possible use
     for this field is to encode an alternate "mix-in" string to be used
     with a string-to-key algorithm (such as is described in section 6.3.2).
     This ability is useful to ease transitions if a realm name needs to
     change (e.g. when a company is acquired); in such a case all existing
     password-derived entries in the KDC database would be flagged as
     needing a special mix-in string until the next password change.
crealm, cname, srealm and sname
     These fields are the same as those described for the ticket in section
     5.3.1.
ticket
     The newly-issued ticket, from section 5.3.1.
enc-part
     This field is a place holder for the ciphertext and related information
     that forms the encrypted part of a message. The description of the
     encrypted part of the message follows each appearance of this field.
```

The encrypted part is encoded as described in section 6.1.

key
This field is the same as described for the ticket in section 5.3.1.

last-req
This field is returned by the KDC and specifies the time(s) of the last request by a principal. Depending on what information is available, this might be the last time that a request for a ticket-granting ticket was made, or the last time that a request based on a ticket-granting ticket was successful. It also might cover all servers for a realm, or just the particular server. Some implementations may display this information to the user to aid in discovering unauthorized use of one's identity. It is similar in spirit to the last login time displayed when logging into timesharing systems.

nonce
This field is described above in section 5.4.1.

key-expiration
The key-expiration field is part of the response from the KDC and specifies the time that the client's secret key is due to expire. The expiration might be the result of password aging or an account expiration. This field will usually be left out of the TGS reply since the response to the TGS request is encrypted in a session key and no client information need be retrieved from the KDC database. It is up to the application client (usually the login program) to take appropriate action (such as notifying the user) if the expiration time is imminent.

flags, authtime, starttime, endtime, renew-till and caddr
These fields are duplicates of those found in the encrypted portion of the attached ticket (see section 5.3.1), provided so the client may verify they match the intended request and to assist in proper ticket caching. If the message is of type KRB_TGS_REP, the caddr field will only be filled in if the request was for a proxy or forwarded ticket, or if the user is substituting a subset of the addresses from the ticket granting ticket. If the client-requested addresses are not present or not used, then the addresses contained in the ticket will be the same as those included in the ticket-granting ticket.

5.5. Client/Server (CS) message specifications

This section specifies the format of the messages used for the authentication of the client to the application server.

5.5.1. KRB_AP_REQ definition

The KRB_AP_REQ message contains the Kerberos protocol version number, the message type KRB_AP_REQ, an options field to indicate any options in use, and the ticket and authenticator themselves. The KRB_AP_REQ message is often referred to as the 'authentication header'.

```
AP-REQ ::=      [APPLICATION 14] SEQUENCE {
                pvno[0]                         INTEGER,
                msg-type[1]                     INTEGER,
                ap-options[2]                   APOptions,
                ticket[3]                       Ticket,
                authenticator[4]                EncryptedData
                                                -- Authenticator from 5.3.2
}

APOptions ::=   BIT STRING {
                reserved(0),
                use-session-key(1),
                mutual-required(2)
```

```
}
```

pvno and msg-type
These fields are described above in section 5.4.1. msg-type is KRB_AP_REQ.

ap-options
This field appears in the application request (KRB_AP_REQ) and affects the way the request is processed. It is a bit-field, where the selected options are indicated by the bit being set (1), and the unselected options and reserved fields being reset (0). The encoding of the bits is specified in section 5.2. The meanings of the options are:

Bit(s)	Name	Description
0	RESERVED	Reserved for future expansion of this field.
1	USE-SESSION-KEY	The USE-SESSION-KEY option indicates that the ticket the client is presenting to a server is encrypted in the session key from the server's ticket-granting ticket. When this option is not specified, the ticket is encrypted in the server's secret key.
2	MUTUAL-REQUIRED	The MUTUAL-REQUIRED option tells the server that the client requires mutual authentication, and that it must respond with a KRB_AP_REP message.
3-31	RESERVED	Reserved for future use.

ticket
This field is a ticket authenticating the client to the server.

authenticator
This contains the authenticator, which includes the client's choice of a subkey. Its encoding is described in section 5.3.2.

5.5.2. KRB_AP_REP definition

The KRB_AP_REP message contains the Kerberos protocol version number, the message type, and an encrypted time- stamp. The message is sent in response to an application request (KRB_AP_REQ) where the mutual authentication option has been selected in the ap-options field.

```
AP-REP ::=          [APPLICATION 15] SEQUENCE {
                    pvno[0]                          INTEGER,
                    msg-type[1]                      INTEGER,
                    enc-part[2]                      EncryptedData
                                                     -- EncAPRepPart
}


EncAPRepPart ::=    [APPLICATION 27[29]] SEQUENCE {
                    ctime[0]                         KerberosTime,
                    cusec[1]                         INTEGER,
                    subkey[2]                        EncryptionKey OPTIONAL,
                    seq-number[3]                    INTEGER OPTIONAL
```

```
}
```

The encoded EncAPRepPart is encrypted in the shared session key of the ticket. The optional subkey field can be used in an application-arranged negotiation to choose a per association session key.

pvno and msg-type
: These fields are described above in section 5.4.1. msg-type is KRB_AP_REP.

enc-part
: This field is described above in section 5.4.2.

ctime
: This field contains the current time on the client's host.

cusec
: This field contains the microsecond part of the client's timestamp.

subkey
: This field contains an encryption key which is to be used to protect this specific application session. See section 3.2.6 for specifics on how this field is used to negotiate a key. Unless an application specifies otherwise, if this field is left out, the sub-session key from the authenticator, or if also left out, the session key from the ticket will be used.

seq-number
: This field is described above in section 5.3.2.

5.5.3. Error message reply

If an error occurs while processing the application request, the KRB_ERROR message will be sent in response. See section 5.9.1 for the format of the error message. The cname and crealm fields may be left out if the server cannot determine their appropriate values from the corresponding KRB_AP_REQ message. If the authenticator was decipherable, the ctime and cusec fields will contain the values from it.

5.6. KRB_SAFE message specification

This section specifies the format of a message that can be used by either side (client or server) of an application to send a tamper-proof message to its peer. It presumes that a session key has previously been exchanged (for example, by using the KRB_AP_REQ/KRB_AP_REP messages).

5.6.1. KRB_SAFE definition

The KRB_SAFE message contains user data along with a collision-proof checksum keyed with the last encryption key negotiated via subkeys, or the session key if no negotiation has occurred. The message fields are:

```
KRB-SAFE ::=        [APPLICATION 20] SEQUENCE {
                    pvno[0]                     INTEGER,
                    msg-type[1]                 INTEGER,
                    safe-body[2]                KRB-SAFE-BODY,
                    cksum[3]                    Checksum
}


KRB-SAFE-BODY ::=   SEQUENCE {
                    user-data[0]                OCTET STRING,
                    timestamp[1]                KerberosTime OPTIONAL,
                    usec[2]                     INTEGER OPTIONAL,
                    seq-number[3]               INTEGER OPTIONAL,
                    s-address[4]                HostAddress OPTIONAL,
                    r-address[5]                HostAddress OPTIONAL
}
```

pvno and msg-type
These fields are described above in section 5.4.1. msg-type is KRB_SAFE.

safe-body
This field is a placeholder for the body of the KRB-SAFE message.

cksum
This field contains the checksum of the application data. Checksum details are described in section 6.4. The checksum is computed over the encoding of the KRB-SAFE sequence. First, the cksum is zeroed and the checksum is computed over the encoding of the KRB-SAFE sequence, then the checksum is set to the result of that computation, and finally the KRB-SAFE sequence is encoded again.

user-data
This field is part of the KRB_SAFE and KRB_PRIV messages and contain the application specific data that is being passed from the sender to the recipient.

timestamp
This field is part of the KRB_SAFE and KRB_PRIV messages. Its contents are the current time as known by the sender of the message. By checking the timestamp, the recipient of the message is able to make sure that it was recently generated, and is not a replay.

usec
This field is part of the KRB_SAFE and KRB_PRIV headers. It contains the microsecond part of the timestamp.

seq-number
This field is described above in section 5.3.2.

s-address
This field specifies the address in use by the sender of the message. It may be omitted if not required by the application protocol. The application designer considering omission of this field is warned, that the inclusion of this address prevents some kinds of replay attacks (e.g., reflection attacks) and that it is only acceptable to omit this address if there is sufficient information in the integrity protected part of the application message for the recipient to unambiguously determine if it was the intended recipient.

r-address
This field specifies the address in use by the recipient of the message. It may be omitted for some uses (such as broadcast protocols), but the recipient may arbitrarily reject such messages. This field along with s-address can be used to help detect messages which have been incorrectly or maliciously delivered to the wrong recipient.

5.7. KRB_PRIV message specification

This section specifies the format of a message that can be used by either side (client or server) of an application to securely and privately send a message to its peer. It presumes that a session key has previously been exchanged (for example, by using the KRB_AP_REQ/KRB_AP_REP messages).

5.7.1. KRB_PRIV definition

The KRB_PRIV message contains user data encrypted in the Session Key. The message fields are:

```
KRB-PRIV ::=        [APPLICATION 21] SEQUENCE {
                    pvno[0]                         INTEGER,
                    msg-type[1]                     INTEGER,
                    enc-part[3]                     EncryptedData
                                                    -- EncKrbPrivPart
}
```

```
EncKrbPrivPart ::=   [APPLICATION 28[31]] SEQUENCE {
                     user-data[0]        OCTET STRING,
                     timestamp[1]        KerberosTime OPTIONAL,
                     usec[2]             INTEGER OPTIONAL,
                     seq-number[3]       INTEGER OPTIONAL,
                    s-address[4]         HostAddress OPTIONAL, -- sender's
                                         addr
                     r-address[5]        HostAddress OPTIONAL -- recip's
                                         addr
}
```

pvno and msg-type
: These fields are described above in section 5.4.1. msg-type is KRB_PRIV.

enc-part
: This field holds an encoding of the EncKrbPrivPart sequence encrypted under the session key[32]. This encrypted encoding is used for the enc-part field of the KRB-PRIV message. See section 6 for the format of the ciphertext.

user-data, timestamp, usec, s-address and r-address
: These fields are described above in section 5.6.1.

seq-number
: This field is described above in section 5.3.2.

5.8. KRB_CRED message specification

This section specifies the format of a message that can be used to send Kerberos credentials from one principal to another. It is presented here to encourage a common mechanism to be used by applications when forwarding tickets or providing proxies to subordinate servers. It presumes that a session key has already been exchanged perhaps by using the KRB_AP_REQ/KRB_AP_REP messages.

5.8.1. KRB_CRED definition

The KRB_CRED message contains a sequence of tickets to be sent and information needed to use the tickets, including the session key from each. The information needed to use the tickets is encrypted under an encryption key previously exchanged or transferred alongside the KRB_CRED message. The message fields are:

```
KRB-CRED         ::= [APPLICATION 22]   SEQUENCE {
                 pvno[0]                INTEGER,
                 msg-type[1]            INTEGER, -- KRB_CRED
                 tickets[2]             SEQUENCE OF Ticket,
                 enc-part[3]            EncryptedData -- EncKrbCredPart
}

EncKrbCredPart   ::= [APPLICATION 29]   SEQUENCE {
                 ticket-info[0]         SEQUENCE OF KrbCredInfo,
                 nonce[1]               INTEGER OPTIONAL,
                 timestamp[2]           KerberosTime OPTIONAL,
                 usec[3]                INTEGER OPTIONAL,
                 s-address[4]           HostAddress OPTIONAL,
                 r-address[5]           HostAddress OPTIONAL
}

KrbCredInfo      ::=                    SEQUENCE {
                 key[0]                 EncryptionKey,
                 prealm[1]              Realm OPTIONAL,
                 pname[2]               PrincipalName OPTIONAL,
                 flags[3]               TicketFlags OPTIONAL,
```

```
                authtime[4]           KerberosTime OPTIONAL,
                starttime[5]          KerberosTime OPTIONAL,
                endtime[6]            KerberosTime OPTIONAL
                renew-till[7]         KerberosTime OPTIONAL,
                srealm[8]             Realm OPTIONAL,
                sname[9]              PrincipalName OPTIONAL,
                caddr[10]             HostAddresses OPTIONAL
}

pvno and msg-type
     These fields are described above in section 5.4.1. msg-type is
     KRB_CRED.
tickets
     These are the tickets obtained from the KDC specifically for use by the
     intended recipient. Successive tickets are paired with the
     corresponding KrbCredInfo sequence from the enc-part of the KRB-CRED
     message.
enc-part
     This field holds an encoding of the EncKrbCredPart sequence encrypted
     under the session key shared between the sender and the intended
     recipient. This encrypted encoding is used for the enc-part field of
     the KRB-CRED message. See section 6 for the format of the ciphertext.
nonce
     If practical, an application may require the inclusion of a nonce
     generated by the recipient of the message. If the same value is
     included as the nonce in the message, it provides evidence that the
     message is fresh and has not been replayed by an attacker. A nonce must
     never be re-used; it should be generated randomly by the recipient of
     the message and provided to the sender of the message in an application
     specific manner.
timestamp and usec
     These fields specify the time that the KRB-CRED message was generated.
     The time is used to provide assurance that the message is fresh.
s-address and r-address
     These fields are described above in section 5.6.1. They are used
     optionally to provide additional assurance of the integrity of the
     KRB-CRED message.
key
     This field exists in the corresponding ticket passed by the KRB-CRED
     message and is used to pass the session key from the sender to the
     intended recipient. The field's encoding is described in section 6.2.

The following fields are optional. If present, they can be associated with
the credentials in the remote ticket file. If left out, then it is assumed
that the recipient of the credentials already knows their value.

prealm and pname
     The name and realm of the delegated principal identity.
flags, authtime, starttime, endtime, renew-till, srealm, sname, and caddr
     These fields contain the values of the corresponding fields from the
     ticket found in the ticket field. Descriptions of the fields are
     identical to the descriptions in the KDC-REP message.

5.9. Error message specification

This section specifies the format for the KRB_ERROR message. The fields
included in the message are intended to return as much information as
possible about an error. It is not expected that all the information
required by the fields will be available for all types of errors. If the
appropriate information is not available when the message is composed, the
corresponding field will be left out of the message.

Note that since the KRB_ERROR message is only optionally integrity
```

protected, it is quite possible for an intruder to synthesize or modify such a message. In particular, this means that unless appropriate integrity protection mechanisms have been applied to the KRB_ERROR message, the client should not use any fields in this message for security-critical purposes, such as setting a system clock or generating a fresh authenticator. The message can be useful, however, for advising a user on the reason for some failure.

5.9.1. KRB_ERROR definition

The KRB_ERROR message consists of the following fields:

```
KRB-ERROR ::=   [APPLICATION 30] SEQUENCE {
                pvno[0]                         INTEGER,
                msg-type[1]                     INTEGER,
                ctime[2]                        KerberosTime OPTIONAL,
                cusec[3]                        INTEGER OPTIONAL,
                stime[4]                        KerberosTime,
                susec[5]                        INTEGER,
                error-code[6]                   INTEGER,
                crealm[7]                       Realm OPTIONAL,
                cname[8]                        PrincipalName OPTIONAL,
                realm[9]                        Realm, -- Correct realm
                sname[10]                       PrincipalName, -- Correct name
                e-text[11]                      GeneralString OPTIONAL,
                e-data[12]                      OCTET STRING OPTIONAL,
                e-cksum[13]                     Checksum OPTIONAL,
}
```

pvno and msg-type
: These fields are described above in section 5.4.1. msg-type is KRB_ERROR.

ctime
: This field is described above in section 5.4.1.

cusec
: This field is described above in section 5.5.2.

stime
: This field contains the current time on the server. It is of type KerberosTime.

susec
: This field contains the microsecond part of the server's timestamp. Its value ranges from 0 to 999999. It appears along with stime. The two fields are used in conjunction to specify a reasonably accurate timestamp.

error-code
: This field contains the error code returned by Kerberos or the server when a request fails. To interpret the value of this field see the list of error codes in section 8. Implementations are encouraged to provide for national language support in the display of error messages.

crealm, cname, srealm and sname
: These fields are described above in section 5.3.1.

e-text
: This field contains additional text to help explain the error code associated with the failed request (for example, it might include a principal name which was unknown).

e-data
: This field contains additional data about the error for use by the application to help it recover from or handle the error. If present, this field will contain the encoding of a sequence of TypedData (TYPED-DATA below), unless the errorcode is KDC_ERR_PREAUTH_REQUIRED,

in which case it will contain the encoding of a sequence of padata fields (METHOD-DATA below), each corresponding to an acceptable pre-authentication method and optionally containing data for the method:

```
TYPED-DATA   ::=   SEQUENCE of TypedData
METHOD-DATA  ::=   SEQUENCE of PA-DATA


TypedData ::=   SEQUENCE {
                    data-type[0]   INTEGER,
                    data-value[1]  OCTET STRING OPTIONAL
}
```

Note that the padata-type field in the PA-DATA structure and the data-type field in the TypedData structure share a common range of allocated values which are coordinated to avoid conflicts. One Kerberos error message, KDC_ERR_PREAUTH_REQUIRED, embeds elements of type PA-DATA, while all other error messages embed TypedData.

While preauthentication methods of type PA-DATA should be encapsulated within a TypedData element of type TD-PADATA, for compatibility with old clients, the KDC should include PA-DATA types below 22 directly as method-data. All new implementations interpreting the METHOD-DATA field for the KDC_ERR_PREAUTH_REQUIRED message must accept a type of TD-PADATA, extract the typed data field and interpret the use any elements encapsulated in the TD-PADATA elements as if they were present in the METHOD-DATA sequence.

Unless otherwise specified, unrecognized TypedData elements within the KRB-ERROR message MAY be ignored by implementations that do not support them. Note that all TypedData MAY be bound to the KRB-ERROR message via the checksum field.

An application may use the TD-APP-DEFINED-ERROR typed data type for data carried in a Kerberos error message that is specific to the application. TD-APP-SPECIFIC must set the data-type value of TypedData to TD-APP-SPECIFIC and the data-value field to

AppSpecificTypedData as follows:

```
    AppSpecificTypedData ::= SEQUENCE {
            oid[0]           OPTIONAL OBJECT IDENTIFIER,
                             -- identifies the application
            data-value[1]    OCTET STRING
                             -- application
                             -- specific data
    }
```

The TD-REQ-NONCE TypedData MAY be used to bind a KRB-ERROR to a KDC-REQ. The data-value is an INTEGER that is equivalent to the nonce in a KDC-REQ.

The TD-REQ-SEQ TypedData MAY be used for binding a KRB-ERROR to the sequence number from an authenticator. The data-value is an INTEGER, and it is identical to sequence number sent in the authenticator.

The data-value for TD-KRB-PRINCIPAL is the Kerberos-defined PrincipalName. The data-value for TD-KRB-REALM is the Kerberos-defined Realm. These TypedData types MAY be used to indicate principal and realm name when appropriate.

e-cksum

This field contains an optional checksum for the KRB-ERROR message. The checksum is calculated over the Kerberos ASN.1 encoding of the KRB-ERROR message with the checksum absent. The checksum is then added to the KRB-ERROR structure and the message is re-encoded. The Checksum should be calculated using the session key from the ticket granting ticket or service ticket, where available. If the error is in response to a TGS or AP request, the checksum should be calculated using the session key from the client's ticket. If the error is in response to an AS request, then the checksum should be calculated using the client's secret key ONLY if there has been suitable preauthentication to prove knowledge of the secret key by the client[33]. If a checksum can not be computed because the key to be used is not available, no checksum will be included.

6. Encryption and Checksum Specifications

This section is undergoing major revision to include rijndael support based on the Internet Draft by Ken Raeburn (draft-raeburn-krb-rijndael-krb-00.txt). The discussions of 3DES are also undergoing revision. Please see http://www.isi.edu/people/bcn/krb-revisions for the latest versions of this section when it becomes available.

7. Naming Constraints

7.1. Realm Names

Although realm names are encoded as GeneralStrings and although a realm can technically select any name it chooses, interoperability across realm boundaries requires agreement on how realm names are to be assigned, and what information they imply.

To enforce these conventions, each realm must conform to the conventions itself, and it must require that any realms with which inter-realm keys are shared also conform to the conventions and require the same from its neighbors.

Kerberos realm names are case sensitive. Realm names that differ only in the case of the characters are not equivalent. There are presently four styles of realm names: domain, X500, other, and reserved. Examples of each style follow:

```
     domain:   ATHENA.MIT.EDU (example)
       X500:   C=US/O=OSF (example)
      other:   NAMETYPE:rest/of.name=without-restrictions (example)
   reserved:   reserved, but will not conflict with above
```

Domain names must look like domain names: they consist of components separated by periods (.) and they contain neither colons (:) nor slashes (/). Though domain names themselves are case insensitive, in order for realms to match, the case must match as well. When establishing a new realm name based on an internet domain name it is recommended by convention that the characters be converted to upper case.

X.500 names contain an equal (=) and cannot contain a colon (:) before the equal. The realm names for X.500 names will be string representations of the names with components separated by slashes. Leading and trailing slashes will not be included. Note that the slash separator is consistent with Kerberos implementations based on RFC1510, but it is different from the separator recommended in RFC2253.

Names that fall into the other category must begin with a prefix that contains no equal (=) or period (.) and the prefix must be followed by a

colon (:) and the rest of the name. All prefixes must be assigned before they may be used. Presently none are assigned.

The reserved category includes strings which do not fall into the first three categories. All names in this category are reserved. It is unlikely that names will be assigned to this category unless there is a very strong argument for not using the 'other' category.

These rules guarantee that there will be no conflicts between the various name styles. The following additional constraints apply to the assignment of realm names in the domain and X.500 categories: the name of a realm for the domain or X.500 formats must either be used by the organization owning (to whom it was assigned) an Internet domain name or X.500 name, or in the case that no such names are registered, authority to use a realm name may be derived from the authority of the parent realm. For example, if there is no domain name for E40.MIT.EDU, then the administrator of the MIT.EDU realm can authorize the creation of a realm with that name.

This is acceptable because the organization to which the parent is assigned is presumably the organization authorized to assign names to its children in the X.500 and domain name systems as well. If the parent assigns a realm name without also registering it in the domain name or X.500 hierarchy, it is the parent's responsibility to make sure that there will not in the future exist a name identical to the realm name of the child unless it is assigned to the same entity as the realm name.

7.2. Principal Names

As was the case for realm names, conventions are needed to ensure that all agree on what information is implied by a principal name. The name-type field that is part of the principal name indicates the kind of information implied by the name. The name-type should be treated as a hint. Ignoring the name type, no two names can be the same (i.e. at least one of the components, or the realm, must be different). The following name types are defined:

name-type	value	meaning
NT-UNKNOWN	0	Name type not known
NT-PRINCIPAL	1	General principal name (e.g. username, or DCE principal)
NT-SRV-INST	2	Service and other unique instance (krbtgt)
NT-SRV-HST	3	Service with host name as instance (telnet, rcommands)
NT-SRV-XHST	4	Service with slash-separated host name components
NT-UID	5	Unique ID
NT-X500-PRINCIPAL	6	Encoded X.509 Distingished name [RFC 1779]
NT-SMTP-NAME	7	Name in form of SMTP email name (e.g. user@foo.com)

When a name implies no information other than its uniqueness at a particular time the name type PRINCIPAL should be used. The principal name type should be used for users, and it might also be used for a unique server. If the name is a unique machine generated ID that is guaranteed never to be reassigned then the name type of UID should be used (note that it is generally a bad idea to reassign names of any type since stale entries might remain in access control lists).

If the first component of a name identifies a service and the remaining components identify an instance of the service in a server specified manner, then the name type of SRV-INST should be used. An example of this name type is the Kerberos ticket-granting service whose name has a first component of krbtgt and a second component identifying the realm for which the ticket is

valid.

If instance is a single component following the service name and the instance identifies the host on which the server is running, then the name type SRV-HST should be used. This type is typically used for Internet services such as telnet and the Berkeley R commands. If the separate components of the host name appear as successive components following the name of the service, then the name type SRV-XHST should be used. This type might be used to identify servers on hosts with X.500 names where the slash (/) might otherwise be ambiguous.

A name type of NT-X500-PRINCIPAL should be used when a name from an X.509 certificate is translated into a Kerberos name. The encoding of the X.509 name as a Kerberos principal shall conform to the encoding rules specified in RFC 2253.

A name type of SMTP allows a name to be of a form that resembles a SMTP email name. This name, including an "@" and a domain name, is used as the one component of the principal name. This name type can be used in conjunction with name-canonicalization to allow a free-form of email address to be specified as a client name and allow the KDC to determine the Kerberos principal name for the requested name. [JBrezak, Raeburn]

A name type of UNKNOWN should be used when the form of the name is not known. When comparing names, a name of type UNKNOWN will match principals authenticated with names of any type. A principal authenticated with a name of type UNKNOWN, however, will only match other names of type UNKNOWN.

Names of any type with an initial component of 'krbtgt' are reserved for the Kerberos ticket granting service. See section 8.2.3 for the form of such names.

7.2.1. Name of server principals

The principal identifier for a server on a host will generally be composed of two parts: (1) the realm of the KDC with which the server is registered, and (2) a two-component name of type NT-SRV-HST if the host name is an Internet domain name or a multi-component name of type NT-SRV-XHST if the name of the host is of a form such as X.500 that allows slash (/) separators. The first component of the two- or multi-component name will identify the service and the latter components will identify the host. Where the name of the host is not case sensitive (for example, with Internet domain names) the name of the host must be lower case. If specified by the application protocol for services such as telnet and the Berkeley R commands which run with system privileges, the first component may be the string 'host' instead of a service specific identifier. When a host has an official name and one or more aliases and the official name can be reliably determined, the official name of the host should be used when constructing the name of the server principal.

8. Constants and other defined values

8.1. Host address types

All negative values for the host address type are reserved for local use. All non-negative values are reserved for officially assigned type fields and interpretations.

The values of the types for the following addresses are chosen to match the defined address family constants in the Berkeley Standard Distributions of Unix. They can be found in with symbolic names AF_xxx (where xxx is an abbreviation of the address family name).

Internet (IPv4) Addresses

Internet (IPv4) addresses are 32-bit (4-octet) quantities, encoded in MSB order. The IPv4 loopback address should not appear in a Kerberos packet. The type of IPv4 addresses is two (2).

Internet (IPv6) Addresses [Westerlund]

IPv6 addresses are 128-bit (16-octet) quantities, encoded in MSB order. The type of IPv6 addresses is twenty-four (24). [RFC1883] [RFC1884]. The following addresses (see [RFC1884]) MUST not appear in any Kerberos packet:

* the Unspecified Address
* the Loopback Address
* Link-Local addresses

IPv4-mapped IPv6 addresses MUST be represented as addresses of type 2.

CHAOSnet addresses

CHAOSnet addresses are 16-bit (2-octet) quantities, encoded in MSB order. The type of CHAOSnet addresses is five (5).

ISO addresses

ISO addresses are variable-length. The type of ISO addresses is seven (7).

Xerox Network Services (XNS) addresses

XNS addresses are 48-bit (6-octet) quantities, encoded in MSB order. The type of XNS addresses is six (6).

AppleTalk Datagram Delivery Protocol (DDP) addresses

AppleTalk DDP addresses consist of an 8-bit node number and a 16-bit network number. The first octet of the address is the node number; the remaining two octets encode the network number in MSB order. The type of AppleTalk DDP addresses is sixteen (16).

DECnet Phase IV addresses

DECnet Phase IV addresses are 16-bit addresses, encoded in LSB order. The type of DECnet Phase IV addresses is twelve (12).

Netbios addresses

Netbios addresses are 16-octet addresses typically composed of 1 to 15 characters, trailing blank (ascii char 20) filled, with a 16th octet of 0x0. The type of Netbios addresses is 20 (0x14).

8.2. KDC messages

8.2.1. UDP/IP transport

When contacting a Kerberos server (KDC) for a KRB_KDC_REQ request using UDP IP transport, the client shall send a UDP datagram containing only an encoding of the request to port 88 (decimal) at the KDC's IP address; the KDC will respond with a reply datagram containing only an encoding of the reply message (either a KRB_ERROR or a KRB_KDC_REP) to the sending port at the sender's IP address. Kerberos servers supporting IP transport must accept UDP requests on port 88 (decimal). The response to a request made

through UDP/IP transport must also use UDP/IP transport.

8.2.2. TCP/IP transport [Westerlund,Danielsson]

Kerberos servers (KDC's) should accept TCP requests on port 88 (decimal) and clients should support the sending of TCP requests on port 88 (decimal). When the KRB_KDC_REQ message is sent to the KDC over a TCP stream, a new connection will be established for each authentication exchange (request and response). The KRB_KDC_REP or KRB_ERROR message will be returned to the client on the same TCP stream that was established for the request. The response to a request made through TCP/IP transport must also use TCP/IP transport. Implementors should note that some extensions to the Kerberos protocol will not work if any implementation not supporting the TCP transport is involved (client or KDC). Implementors are strongly urged to support the TCP transport on both the client and server and are advised that the current notation of "should" support will likely change in the future to must support. The KDC may close the TCP stream after sending a response, but may leave the stream open if it expects a followup - in which case it may close the stream at any time if resource constraints or other factors make it desirable to do so. Care must be taken in managing TCP/IP connections with the KDC to prevent denial of service attacks based on the number of TCP/IP connections with the KDC that remain open. If multiple exchanges with the KDC are needed for certain forms of preauthentication, multiple TCP connections may be required. A client may close the stream after receiving response, and should close the stream if it does not expect to send followup messages. The client must be prepared to have the stream closed by the KDC at anytime, in which case it must simply connect again when it is ready to send subsequent messages.

The first four octets of the TCP stream used to transmit the request will encode in network byte order the length of the request (KRB_KDC_REQ), and the length will be followed by the request itself. The response will similarly be preceded by a 4 octet encoding in network byte order of the length of the KRB_KDC_REP or the KRB_ERROR message and will be followed by the KRB_KDC_REP or the KRB_ERROR response. If the sign bit is set on the integer represented by the first 4 octets, then the next 4 octets will be read, extending the length of the field by another 4 octets (less the sign bit of the additional four octets which is reserved for future expansion and which at present must be zero).

8.2.3. OSI transport

During authentication of an OSI client to an OSI server, the mutual authentication of an OSI server to an OSI client, the transfer of credentials from an OSI client to an OSI server, or during exchange of private or integrity checked messages, Kerberos protocol messages may be treated as opaque objects and the type of the authentication mechanism will be:

OBJECT IDENTIFIER ::= {iso (1), org(3), dod(6),internet(1), security(5),kerberosv5(2)}

Depending on the situation, the opaque object will be an authentication header (KRB_AP_REQ), an authentication reply (KRB_AP_REP), a safe message (KRB_SAFE), a private message (KRB_PRIV), or a credentials message (KRB_CRED). The opaque data contains an application code as specified in the ASN.1 description for each message. The application code may be used by Kerberos to determine the message type.

8.2.3. Name of the TGS

The principal identifier of the ticket-granting service shall be composed of

three parts: (1) the realm of the KDC issuing the TGS ticket (2) a two-part name of type NT-SRV-INST, with the first part "krbtgt" and the second part the name of the realm which will accept the ticket-granting ticket. For example, a ticket-granting ticket issued by the ATHENA.MIT.EDU realm to be used to get tickets from the ATHENA.MIT.EDU KDC has a principal identifier of "ATHENA.MIT.EDU" (realm), ("krbtgt", "ATHENA.MIT.EDU") (name). A ticket-granting ticket issued by the ATHENA.MIT.EDU realm to be used to get tickets from the MIT.EDU realm has a principal identifier of "ATHENA.MIT.EDU" (realm), ("krbtgt", "MIT.EDU") (name).

8.3. Protocol constants and associated values

The following tables list constants used in the protocol and define their meanings. Ranges are specified in the "specification" section that limit the values of constants for which values are defined here. This allows implementations to make assumptions about the maximum values that will be received for these constants. Implementation receiving values outside the range specified in the "specification" section may reject the request, but they must recover cleanly.

Encryption type	etype value	block size	minimum pad size	confounder size
NULL	0	1	0	0
des-cbc-crc	1	8	4	8
des-cbc-md4	2	8	0	8
des-cbc-md5	3	8	0	8
[reserved]	4			
des3-cbc-md5	5	8	0	8
[reserved]	6			
des3-cbc-sha1	7	8	0	8
dsaWithSHA1-CmsOID	9			(pkinit)
md5WithRSAEncryption-CmsOID	10			(pkinit)
sha1WithRSAEncryption-CmsOID	11			(pkinit)
rc2CBC-EnvOID	12			(pkinit)
rsaEncryption-EnvOID	13			(pkinit from PKCS#1 v1.5)
rsaES-OAEP-ENV-OID	14			(pkinit from PKCS#1 v2.0)
des-ede3-cbc-Env-OID	15			(pkinit)
des3-cbc-sha1-kd	16			(Tom Yu)
rc4-hmac	23			(swift)
rc4-hmac-exp	24			(swift)
subkey-keynaterial	65			(opaque mhur)

[reserved] 0x8003

Checksum type	sumtype value	checksum size
CRC32	1	4
rsa-md4	2	16
rsa-md4-des	3	24
des-mac	4	16
des-mac-k	5	8
rsa-md4-des-k	6	16 (drop rsa ?)
rsa-md5	7	16 (drop rsa ?)
rsa-md5-des	8	24 (drop rsa ?)
rsa-md5-des3	9	24 (drop rsa ?)
hmac-sha1-des3-kd	12	20
hmac-sha1-des3	13	20
sha1 (unkeyed)	14	20

padata and data types	padata-type value	comment
PA-TGS-REQ	1	
PA-ENC-TIMESTAMP	2	
PA-PW-SALT	3	

```
[reserved]                      4
PA-ENC-UNIX-TIME                5                  (depricated)
PA-SANDIA-SECUREID              6
PA-SESAME                       7
PA-OSF-DCE                      8
PA-CYBERSAFE-SECUREID           9
PA-AFS3-SALT                    10
PA-ETYPE-INFO                   11
PA-SAM-CHALLENGE                12                  (sam/otp)
PA-SAM-RESPONSE                 13                  (sam/otp)
PA-PK-AS-REQ                    14                  (pkinit)
PA-PK-AS-REP                    15                  (pkinit)
PA-USE-SPECIFIED-KVNO           20
PA-SAM-REDIRECT                 21                  (sam/otp)
PA-GET-FROM-TYPED-DATA          22                  (embedded in typed data)
TD-PADATA                       22                  (embeds padata)
PA-SAM-ETYPE-INFO               23                  (sam/otp)
TD-PKINIT-CMS-CERTIFICATES      101      CertificateSet from CMS
TD-KRB-PRINCIPAL                102      PrincipalName (see Sec.5.9.1)
TD-KRB-REALM                    103      Realm (see Sec.5.9.1)
TD-TRUSTED-CERTIFIERS           104      from PKINIT
TD-CERTIFICATE-INDEX            105      from PKINIT
TD-APP-DEFINED-ERROR            106      application specific (see Sec.5.9.1)
TD-REQ-NONCE                    107      INTEGER (see Sec.5.9.1)
TD-REQ-SEQ                      108      INTEGER (see Sec.5.9.1)

authorization data type         ad-type value
AD-IF-RELEVANT                     1
AD-INTENDED-FOR-SERVER             2
AD-INTENDED-FOR-APPLICATION-CLASS  3
AD-KDC-ISSUED                      4
AD-OR                              5
AD-MANDATORY-TICKET-EXTENSIONS     6
AD-IN-TICKET-EXTENSIONS            7
reserved values                    8-63
OSF-DCE                            64
SESAME                             65
AD-OSF-DCE-PKI-CERTID              66         (hemsath@us.ibm.com)
AD-WIN200-PAC                     128         (jbrezak@exchange.microsoft.com)


Ticket Extension Types

TE-TYPE-NULL                  0      Null ticket extension
TE-TYPE-EXTERNAL-ADATA        1      Integrity protected authorization data
[reserved]                    2      TE-TYPE-PKCROSS-KDC  (I have reservations)
TE-TYPE-PKCROSS-CLIENT        3      PKCROSS cross realm key ticket
TE-TYPE-CYBERSAFE-EXT         4      Assigned to CyberSafe Corp
[reserved]                    5      TE-TYPE-DEST-HOST (I have reservations)

alternate authentication type   method-type value
reserved values                 0-63
ATT-CHALLENGE-RESPONSE          64

transited encoding type         tr-type value
DOMAIN-X500-COMPRESS            1
reserved values                 all others

Label               Value   Meaning or MIT code

pvno                    5   current Kerberos protocol version number

message types
```

```
KRB_AS_REQ             10   Request for initial authentication
KRB_AS_REP             11   Response to KRB_AS_REQ request
KRB_TGS_REQ            12   Request for authentication based on TGT
KRB_TGS_REP            13   Response to KRB_TGS_REQ request
KRB_AP_REQ             14   application request to server
KRB_AP_REP             15   Response to KRB_AP_REQ_MUTUAL
KRB_SAFE               20   Safe (checksummed) application message
KRB_PRIV               21   Private (encrypted) application message
KRB_CRED               22   Private (encrypted) message to forward credentials
KRB_ERROR              30   Error response

name types

KRB_NT_UNKNOWN        0  Name type not known
KRB_NT_PRINCIPAL      1  Just the name of the principal as in DCE, or for users
KRB_NT_SRV_INST       2  Service and other unique instance (krbtgt)
KRB_NT_SRV_HST        3  Service with host name as instance (telnet, rcommands)
KRB_NT_SRV_XHST       4  Service with host as remaining components
KRB_NT_UID            5  Unique ID
KRB_NT_X500_PRINCIPAL 6  Encoded X.509 Distingished name [RFC 2253]


error codes

KDC_ERR_NONE                   0   No error
KDC_ERR_NAME_EXP               1   Client's entry in database has expired
KDC_ERR_SERVICE_EXP            2   Server's entry in database has expired
KDC_ERR_BAD_PVNO               3   Requested protocol version number not
supported
KDC_ERR_C_OLD_MAST_KVNO        4   Client's key encrypted in old master key
KDC_ERR_S_OLD_MAST_KVNO        5   Server's key encrypted in old master key
KDC_ERR_C_PRINCIPAL_UNKNOWN    6   Client not found in Kerberos database
KDC_ERR_S_PRINCIPAL_UNKNOWN    7   Server not found in Kerberos database
KDC_ERR_PRINCIPAL_NOT_UNIQUE   8   Multiple principal entries in database
KDC_ERR_NULL_KEY               9   The client or server has a null key
KDC_ERR_CANNOT_POSTDATE       10   Ticket not eligible for postdating
KDC_ERR_NEVER_VALID           11   Requested start time is later than end time
KDC_ERR_POLICY                12   KDC policy rejects request
KDC_ERR_BADOPTION             13   KDC cannot accommodate requested option
KDC_ERR_ETYPE_NOSUPP          14   KDC has no support for encryption type
KDC_ERR_SUMTYPE_NOSUPP        15   KDC has no support for checksum type
KDC_ERR_PADATA_TYPE_NOSUPP    16   KDC has no support for padata type
KDC_ERR_TRTYPE_NOSUPP         17   KDC has no support for transited type
KDC_ERR_CLIENT_REVOKED        18   Clients credentials have been revoked
KDC_ERR_SERVICE_REVOKED       19   Credentials for server have been revoked
KDC_ERR_TGT_REVOKED           20   TGT has been revoked
KDC_ERR_CLIENT_NOTYET         21   Client not yet valid - try again later
KDC_ERR_SERVICE_NOTYET        22   Server not yet valid - try again later
KDC_ERR_KEY_EXPIRED           23   Password has expired - change password to
reset
KDC_ERR_PREAUTH_FAILED        24   Pre-authentication information was invalid
KDC_ERR_PREAUTH_REQUIRED      25   Additional pre-authenticationrequired [40]
KDC_ERR_SERVER_NOMATCH        26   Requested server and ticket don't match
KDC_ERR_MUST_USE_USER2USER    27   Server principal valid for user2user only
KDC_ERR_PATH_NOT_ACCPETED     28   KDC Policy rejects transited path
KDC_ERR_SVC_UNAVAILABLE       29   A service is not available
KRB_AP_ERR_BAD_INTEGRITY      31   Integrity check on decrypted field failed
KRB_AP_ERR_TKT_EXPIRED        32   Ticket expired
KRB_AP_ERR_TKT_NYV            33   Ticket not yet valid
KRB_AP_ERR_REPEAT             34   Request is a replay
KRB_AP_ERR_NOT_US             35   The ticket isn't for us
KRB_AP_ERR_BADMATCH           36   Ticket and authenticator don't match
```

```
KRB_AP_ERR_SKEW                 37   Clock skew too great
KRB_AP_ERR_BADADDR              38   Incorrect net address
KRB_AP_ERR_BADVERSION           39   Protocol version mismatch
KRB_AP_ERR_MSG_TYPE             40   Invalid msg type
KRB_AP_ERR_MODIFIED             41   Message stream modified
KRB_AP_ERR_BADORDER             42   Message out of order
KRB_AP_ERR_BADKEYVER            44   Specified version of key is not available
KRB_AP_ERR_NOKEY                45   Service key not available
KRB_AP_ERR_MUT_FAIL             46   Mutual authentication failed
KRB_AP_ERR_BADDIRECTION         47   Incorrect message direction
KRB_AP_ERR_METHOD               48   Alternative authentication method required
KRB_AP_ERR_BADSEQ               49   Incorrect sequence number in message
KRB_AP_ERR_INAPP_CKSUM          50   Inappropriate type of checksum in message
KRB_AP_PATH_NOT_ACCEPTED        51   Policy rejects transited path
KRB_ERR_RESPONSE_TOO_BIG        52   Response too big for UDP, retry with TCP
KRB_ERR_GENERIC                 60   Generic error (description in e-text)
KRB_ERR_FIELD_TOOLONG           61   Field is too long for this implementation
KDC_ERROR_CLIENT_NOT_TRUSTED         62 (pkinit)
KDC_ERROR_KDC_NOT_TRUSTED            63 (pkinit)


KDC_ERROR_INVALID_SIG                64 (pkinit)
KDC_ERR_KEY_TOO_WEAK                 65 (pkinit)
KDC_ERR_CERTIFICATE_MISMATCH         66 (pkinit)
KRB_AP_ERR_NO_TGT                    67 (user-to-user)
KDC_ERR_WRONG_REALM                  68 (user-to-user)
KRB_AP_ERR_USER_TO_USER_REQUIRED     69 (user-to-user)
KDC_ERR_CANT_VERIFY_CERTIFICATE      70 (pkinit)
KDC_ERR_INVALID_CERTIFICATE          71 (pkinit)
KDC_ERR_REVOKED_CERTIFICATE          72 (pkinit)
KDC_ERR_REVOCATION_STATUS_UNKNOWN    73 (pkinit)
KDC_ERR_REVOCATION_STATUS_UNAVAILABLE 74 (pkinit)
KDC_ERR_CLIENT_NAME_MISMATCH         75 (pkinit)
KDC_ERR_KDC_NAME_MISMATCH            76 (pkinit)
```

9. Interoperability requirements

Version 5 of the Kerberos protocol supports a myriad of options. Among these are multiple encryption and checksum types, alternative encoding schemes for the transited field, optional mechanisms for pre-authentication, the handling of tickets with no addresses, options for mutual authentication, user to user authentication, support for proxies, forwarding, postdating, and renewing tickets, the format of realm names, and the handling of authorization data.

In order to ensure the interoperability of realms, it is necessary to define a minimal configuration which must be supported by all implementations. This minimal configuration is subject to change as technology does. For example, if at some later date it is discovered that one of the required encryption or checksum algorithms is not secure, it will be replaced.

9.1. Specification 2

This section defines the second specification of these options. Implementations which are configured in this way can be said to support Kerberos Version 5 Specification 2 (5.1). Specification 1 (deprecated) may be found in RFC1510.

Transport

TCP/IP and UDP/IP transport must be supported by KDCs claiming conformance to specification 2. Kerberos clients claiming conformance to specification 2

must support UDP/IP transport for messages with the KDC and should support TCP/IP transport.

Encryption and checksum methods

The following encryption and checksum mechanisms must be supported. Implementations may support other mechanisms as well, but the additional mechanisms may only be used when communicating with principals known to also support them: This list is to be determined.

Encryption: DES-CBC-MD5, DES3-CBC-SHA1-KD, RIJNDAEL(decide identifier)
Checksums: CRC-32, DES-MAC, DES-MAC-K, DES-MD5, HMAC-SHA1-DES3-KD

Realm Names

All implementations must understand hierarchical realms in both the Internet Domain and the X.500 style. When a ticket granting ticket for an unknown realm is requested, the KDC must be able to determine the names of the intermediate realms between the KDCs realm and the requested realm.

Transited field encoding

DOMAIN-X500-COMPRESS (described in section 3.3.3.2) must be supported. Alternative encodings may be supported, but they may be used only when that encoding is supported by ALL intermediate realms.

Pre-authentication methods

The TGS-REQ method must be supported. The TGS-REQ method is not used on the initial request. The PA-ENC-TIMESTAMP method must be supported by clients but whether it is enabled by default may be determined on a realm by realm basis. If not used in the initial request and the error KDC_ERR_PREAUTH_REQUIRED is returned specifying PA-ENC-TIMESTAMP as an acceptable method, the client should retry the initial request using the PA-ENC-TIMESTAMP preauthentication method. Servers need not support the PA-ENC-TIMESTAMP method, but if not supported the server should ignore the presence of PA-ENC-TIMESTAMP pre-authentication in a request.

Mutual authentication

Mutual authentication (via the KRB_AP_REP message) must be supported.

Ticket addresses and flags

All KDC's must pass through tickets that carry no addresses (i.e. if a TGT contains no addresses, the KDC will return derivative tickets), but each realm may set its own policy for issuing such tickets, and each application server will set its own policy with respect to accepting them.

Proxies and forwarded tickets must be supported. Individual realms and application servers can set their own policy on when such tickets will be accepted.

All implementations must recognize renewable and postdated tickets, but need not actually implement them. If these options are not supported, the starttime and endtime in the ticket shall specify a ticket's entire useful life. When a postdated ticket is decoded by a server, all implementations shall make the presence of the postdated flag visible to the calling server.

User-to-user authentication

Support for user to user authentication (via the ENC-TKT-IN-SKEY KDC option)

must be provided by implementations, but individual realms may decide as a matter of policy to reject such requests on a per-principal or realm-wide basis.

Authorization data

Implementations must pass all authorization data subfields from ticket-granting tickets to any derivative tickets unless directed to suppress a subfield as part of the definition of that registered subfield type (it is never incorrect to pass on a subfield, and no registered subfield types presently specify suppression at the KDC).

Implementations must make the contents of any authorization data subfields available to the server when a ticket is used. Implementations are not required to allow clients to specify the contents of the authorization data fields.

Constant ranges

All protocol constants are constrained to 32 bit (signed) values unless further constrained by the protocol definition. This limit is provided to allow implementations to make assumptions about the maximum values that will be received for these constants. Implementation receiving values outside this range may reject the request, but they must recover cleanly.

9.2. Recommended KDC values

Following is a list of recommended values for a KDC implementation, based on the list of suggested configuration constants (see section 4.4).

minimum lifetime	5 minutes
maximum renewable lifetime	1 week
maximum ticket lifetime	1 day
empty addresses	only when suitable restrictions appear in authorization data
proxiable, etc.	Allowed.

10. REFERENCES

[NT94] B. Clifford Neuman and Theodore Y. Ts'o, "An Authentication Service for Computer Networks," IEEE Communications Magazine, Vol. 32(9), pp. 33-38 (September 1994).

[MNSS87] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, Section E.2.1: Kerberos Authentication and Authorization System, M.I.T. Project Athena, Cambridge, Massachusetts (December 21, 1987).

[SNS88] J. G. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," pp. 191-202 in Usenix Conference Proceedings, Dallas, Texas (February, 1988).

[NS78] Roger M. Needham and Michael D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," Communications of the ACM, Vol. 21(12), pp. 993-999 (December, 1978).

[DS81] Dorothy E. Denning and Giovanni Maria Sacco, "Timestamps in Key Distribution Protocols," Communications of the ACM, Vol. 24(8), pp. 533-536 (August 1981).

[KNT92] John T. Kohl, B. Clifford Neuman, and Theodore Y. Ts'o,

"The Evolution of the Kerberos Authentication Service," in an IEEE Computer Society Text soon to be published (June 1992).

[Neu93] B. Clifford Neuman, "Proxy-Based Authorization and Accounting for Distributed Systems," in Proceedings of the 13th International Conference on Distributed Computing Systems, Pittsburgh, PA (May, 1993).

[DS90] Don Davis and Ralph Swick, "Workstation Services and Kerberos Authentication at Project Athena," Technical Memorandum TM-424, MIT Laboratory for Computer Science (February 1990).

[LGDSR87] P. J. Levine, M. R. Gretzinger, J. M. Diaz, W. E. Sommerfeld, and K. Raeburn, Section E.1: Service Management System, M.I.T. Project Athena, Cambridge, Massachusetts (1987).

[X509-88] CCITT, Recommendation X.509: The Directory Authentication Framework, December 1988.

[Pat92]. J. Pato, Using Pre-Authentication to Avoid Password Guessing Attacks, Open Software Foundation DCE Request for Comments 26 (December 1992).

[DES77] National Bureau of Standards, U.S. Department of Commerce, "Data Encryption Standard," Federal Information Processing Standards Publication 46, Washington, DC (1977).

[DESM80] National Bureau of Standards, U.S. Department of Commerce, "DES Modes of Operation," Federal Information Processing Standards Publication 81, Springfield, VA (December 1980).

[SG92] Stuart G. Stubblebine and Virgil D. Gligor, "On Message Integrity in Cryptographic Protocols," in Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, California (May 1992).

[IS3309] International Organization for Standardization, "ISO Information Processing Systems - Data Communication - High-Level Data Link Control Procedure - Frame Structure," IS 3309 (October 1984). 3rd Edition.

[MD4-92] R. Rivest, "The MD4 Message Digest Algorithm," RFC 1320, MIT Laboratory for Computer Science (April 1992).

[MD5-92] R. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, MIT Laboratory for Computer Science (April 1992).

[KBC96] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," Working Draft draft-ietf-ipsec-hmac-md5-01.txt, (August 1996).

[Horowitz96] Horowitz, M., "Key Derivation for Authentication, Integrity, and Privacy", draft-horowitz-key-derivation-02.txt, August 1998.

[HorowitzB96] Horowitz, M., "Key Derivation for Kerberos V5", draft-
horowitz-kerb-key-derivation-01.txt, September 1998.

[Krawczyk96] Krawczyk, H., Bellare, and M., Canetti, R., "HMAC:
Keyed-Hashing for Message Authentication", =
draft-ietf-ipsec-hmac-
md5-01.txt, August, 1996.

A. Pseudo-code for protocol processing

This appendix provides pseudo-code describing how the messages are to be constructed and interpreted by clients and servers.

A.1. KRB_AS_REQ generation

```
        request.pvno :=3D protocol version; /* pvno =3D 5 */
        request.msg-type :=3D message type; /* type =3D KRB_AS_REQ */

        if(pa_enc_timestamp_required) then
                request.padata.padata-type =3D PA-ENC-TIMESTAMP;
                get system_time;
                padata-body.patimestamp,pausec =3D system_time;
                encrypt padata-body into request.padata.padata-value
                        using client.key; /* derived from password */
        endif

        body.kdc-options :=3D users's preferences;
        body.cname :=3D user's name;
        body.realm :=3D user's realm;
        body.sname :=3D service's name; /* usually "krbtgt",  =
"localrealm" */
        if (body.kdc-options.POSTDATED is set) then
                body.from :=3D requested starting time;
        else
                omit body.from;
        endif
        body.till :=3D requested end time;
        if (body.kdc-options.RENEWABLE is set) then
                body.rtime :=3D requested final renewal time;
        endif
        body.nonce :=3D random_nonce();
        body.etype :=3D requested etypes;
        if (user supplied addresses) then
                body.addresses :=3D user's addresses;
        else
                omit body.addresses;
        endif
        omit body.enc-authorization-data;
        request.req-body :=3D body;

        kerberos :=3D lookup(name of local kerberos server (or =
servers));
        send(packet,kerberos);

        wait(for response);
        if (timed_out) then
                retry or use alternate server;
        endif
```

A.2. KRB_AS_REQ verification and KRB_AS_REP generation

```
        decode message into req;

        client :=3D lookup(req.cname,req.realm);
        server :=3D lookup(req.sname,req.realm);


        get system_time;
        kdc_time :=3D system_time.seconds;

        if (!client) then
                /* no client in Database */
                error_out(KDC_ERR_C_PRINCIPAL_UNKNOWN);
        endif
        if (!server) then
                /* no server in Database */
                error_out(KDC_ERR_S_PRINCIPAL_UNKNOWN);
        endif

        if(client.pa_enc_timestamp_required and
           pa_enc_timestamp not present) then
                error_out(KDC_ERR_PREAUTH_REQUIRED(PA_ENC_TIMESTAMP));
        endif

        if(pa_enc_timestamp present) then
                decrypt req.padata-value into decrypted_enc_timestamp
                        using client.key;
                        using auth_hdr.authenticator.subkey;
                if (decrypt_error()) then
                        error_out(KRB_AP_ERR_BAD_INTEGRITY);
                if(decrypted_enc_timestamp is not within allowable skew) =
then
                        error_out(KDC_ERR_PREAUTH_FAILED);
                endif
                if(decrypted_enc_timestamp and usec is replay)
                        error_out(KDC_ERR_PREAUTH_FAILED);
                endif
                add decrypted_enc_timestamp and usec to replay cache;
        endif

        use_etype :=3D first supported etype in req.etypes;

        if (no support for req.etypes) then
                error_out(KDC_ERR_ETYPE_NOSUPP);
        endif

        new_tkt.vno :=3D ticket version; /* =3D 5 */
        new_tkt.sname :=3D req.sname;
        new_tkt.srealm :=3D req.srealm;
        reset all flags in new_tkt.flags;

        /* It should be noted that local policy may affect the  */
        /* processing of any of these flags.  For example, some */
        /* realms may refuse to issue renewable tickets         */

        if (req.kdc-options.FORWARDABLE is set) then
                set new_tkt.flags.FORWARDABLE;
        endif
        if (req.kdc-options.PROXIABLE is set) then
                set new_tkt.flags.PROXIABLE;
        endif

        if (req.kdc-options.ALLOW-POSTDATE is set) then
                set new_tkt.flags.MAY-POSTDATE;
```

```
        endif


        if ((req.kdc-options.RENEW is set) or
            (req.kdc-options.VALIDATE is set) or
            (req.kdc-options.PROXY is set) or
            (req.kdc-options.FORWARDED is set) or
            (req.kdc-options.ENC-TKT-IN-SKEY is set)) then
                error_out(KDC_ERR_BADOPTION);
        endif

        new_tkt.session :=3D random_session_key();
        new_tkt.cname :=3D req.cname;
        new_tkt.crealm :=3D req.crealm;
        new_tkt.transited :=3D empty_transited_field();

        new_tkt.authtime :=3D kdc_time;

        if (req.kdc-options.POSTDATED is set) then
           if (against_postdate_policy(req.from)) then
                error_out(KDC_ERR_POLICY);
           endif
           set new_tkt.flags.POSTDATED;
           set new_tkt.flags.INVALID;
           new_tkt.starttime :=3D req.from;
        else
           omit new_tkt.starttime; /* treated as authtime when omitted =
*/
        endif
        if (req.till =3D 0) then
                till :=3D infinity;
        else
                till :=3D req.till;
        endif

        new_tkt.endtime :=3D min(till,
                             new_tkt.starttime+client.max_life,
                             new_tkt.starttime+server.max_life,
                             new_tkt.starttime+max_life_for_realm);

        if ((req.kdc-options.RENEWABLE-OK is set) and
            (new_tkt.endtime < req.till)) then
                /* we set the RENEWABLE option for later processing */
                set req.kdc-options.RENEWABLE;
                req.rtime :=3D req.till;
        endif

        if (req.rtime =3D 0) then
                rtime :=3D infinity;
        else
                rtime :=3D req.rtime;
        endif

        if (req.kdc-options.RENEWABLE is set) then
                set new_tkt.flags.RENEWABLE;
                new_tkt.renew-till :=3D min(rtime,
                                        new_tkt.starttime+client.max_rlife,
                                        new_tkt.starttime+server.max_rlife,

new_tkt.starttime+max_rlife_for_realm);


        else
```

```
                omit new_tkt.renew-till; /* only present if RENEWABLE */
        endif

        if (req.addresses) then
                new_tkt.caddr :=3D req.addresses;
        else
                omit new_tkt.caddr;
        endif

        new_tkt.authorization_data :=3D empty_authorization_data();

        encode to-be-encrypted part of ticket into OCTET STRING;
        new_tkt.enc-part :=3D encrypt OCTET STRING
                using etype_for_key(server.key), server.key, =
server.p_kvno;

        /* Start processing the response */

        resp.pvno :=3D 5;
        resp.msg-type :=3D KRB_AS_REP;
        resp.cname :=3D req.cname;
        resp.crealm :=3D req.realm;
        resp.ticket :=3D new_tkt;

        resp.key :=3D new_tkt.session;
        resp.last-req :=3D fetch_last_request_info(client);
        resp.nonce :=3D req.nonce;
        resp.key-expiration :=3D client.expiration;
        resp.flags :=3D new_tkt.flags;

        resp.authtime :=3D new_tkt.authtime;
        resp.starttime :=3D new_tkt.starttime;
        resp.endtime :=3D new_tkt.endtime;

        if (new_tkt.flags.RENEWABLE) then
                resp.renew-till :=3D new_tkt.renew-till;
        endif

        resp.realm :=3D new_tkt.realm;
        resp.sname :=3D new_tkt.sname;

        resp.caddr :=3D new_tkt.caddr;

        encode body of reply into OCTET STRING;

        resp.enc-part :=3D encrypt OCTET STRING
                        using use_etype, client.key, client.p_kvno;
        send(resp);
```

A.3. KRB_AS_REP verification

```
        decode response into resp;

        if (resp.msg-type =3D KRB_ERROR) then
                if(error =3D KDC_ERR_PREAUTH_REQUIRED(PA_ENC_TIMESTAMP)) =
then
                        set pa_enc_timestamp_required;
                        goto KRB_AS_REQ;
                endif
                process_error(resp);
                return;
        endif
```

```
        /* On error, discard the response, and zero the session key */
        /* from the response immediately */

        key =3D get_decryption_key(resp.enc-part.kvno, =
resp.enc-part.etype,
                                  resp.padata);
        unencrypted part of resp :=3D decode of decrypt of resp.enc-part
                                 using resp.enc-part.etype and key;
        zero(key);

        if (common_as_rep_tgs_rep_checks fail) then
                destroy resp.key;
                return error;
        endif

        if near(resp.princ_exp) then
                print(warning message);
        endif
        save_for_later(ticket,session,client,server,times,flags);
```

A.4. KRB_AS_REP and KRB_TGS_REP common checks

```
        if (decryption_error() or
            (req.cname !=3D resp.cname) or
            (req.realm !=3D resp.crealm) or
            (req.sname !=3D resp.sname) or
            (req.realm !=3D resp.realm) or
            (req.nonce !=3D resp.nonce) or
            (req.addresses !=3D resp.caddr)) then
                destroy resp.key;
                return KRB_AP_ERR_MODIFIED;
        endif

        /* make sure no flags are set that shouldn't be, and that all =
that */
        /* should be are set                                          =
  */
        if (!check_flags_for_compatability(req.kdc-options,resp.flags)) =
then
                destroy resp.key;
                return KRB_AP_ERR_MODIFIED;
        endif


        if ((req.from =3D 0) and
            (resp.starttime is not within allowable skew)) then
                destroy resp.key;
                return KRB_AP_ERR_SKEW;
        endif
        if ((req.from !=3D 0) and (req.from !=3D resp.starttime)) then
                destroy resp.key;
                return KRB_AP_ERR_MODIFIED;
        endif
        if ((req.till !=3D 0) and (resp.endtime > req.till)) then
                destroy resp.key;
                return KRB_AP_ERR_MODIFIED;
        endif

        if ((req.kdc-options.RENEWABLE is set) and
            (req.rtime !=3D 0) and (resp.renew-till > req.rtime)) then
                destroy resp.key;
                return KRB_AP_ERR_MODIFIED;
```

```
        endif
        if ((req.kdc-options.RENEWABLE-OK is set) and
            (resp.flags.RENEWABLE) and
            (req.till !=3D 0) and
            (resp.renew-till > req.till)) then
                destroy resp.key;
                return KRB_AP_ERR_MODIFIED;
        endif
```

A.5. KRB_TGS_REQ generation

```
        /* Note that make_application_request might have to recursivly   =
  */
        /* call this routine to get the appropriate ticket-granting =
ticket */

        request.pvno :=3D protocol version; /* pvno =3D 5 */
        request.msg-type :=3D message type; /* type =3D KRB_TGS_REQ */

        body.kdc-options :=3D users's preferences;
        /* If the TGT is not for the realm of the end-server  */
        /* then the sname will be for a TGT for the end-realm */
        /* and the realm of the requested ticket (body.realm) */
        /* will be that of the TGS to which the TGT we are    */
        /* sending applies                                    */
        body.sname :=3D service's name;
        body.realm :=3D service's realm;

        if (body.kdc-options.POSTDATED is set) then
                body.from :=3D requested starting time;
        else
                omit body.from;
        endif
        body.till :=3D requested end time;
        if (body.kdc-options.RENEWABLE is set) then
                body.rtime :=3D requested final renewal time;
        endif


        body.nonce :=3D random_nonce();
        body.etype :=3D requested etypes;
        if (user supplied addresses) then
                body.addresses :=3D user's addresses;
        else
                omit body.addresses;
        endif

        body.enc-authorization-data :=3D user-supplied data;
        if (body.kdc-options.ENC-TKT-IN-SKEY) then
                body.additional-tickets_ticket :=3D second TGT;
        endif

        request.req-body :=3D body;
        check :=3D generate_checksum (req.body,checksumtype);

        request.padata[0].padata-type :=3D PA-TGS-REQ;
        request.padata[0].padata-value :=3D create a KRB_AP_REQ using
                                    the TGT and checksum

        /* add in any other padata as required/supplied */

        kerberos :=3D lookup(name of local kerberose server (or =
servers));
```

```
        send(packet,kerberos);

        wait(for response);
        if (timed_out) then
                retry or use alternate server;
        endif
```

A.6. KRB_TGS_REQ verification and KRB_TGS_REP generation

```
        /* note that reading the application request requires first
        determining the server for which a ticket was issued, and =
choosing the
        correct key for decryption.  The name of the server appears in =
the
        plaintext part of the ticket. */

        if (no KRB_AP_REQ in req.padata) then
                error_out(KDC_ERR_PADATA_TYPE_NOSUPP);
        endif
        verify KRB_AP_REQ in req.padata;

        /* Note that the realm in which the Kerberos server is operating =
is
        determined by the instance from the ticket-granting ticket.  The =
realm
        in the ticket-granting ticket is the realm under which the =
ticket
        granting ticket was issued.  It is possible for a single =
Kerberos
        server to support more than one realm. */

        auth_hdr :=3D KRB_AP_REQ;
        tgt :=3D auth_hdr.ticket;


        if (tgt.sname is not a TGT for local realm and is not req.sname) =
then
                error_out(KRB_AP_ERR_NOT_US);

        realm :=3D realm_tgt_is_for(tgt);

        decode remainder of request;

        if (auth_hdr.authenticator.cksum is missing) then
                error_out(KRB_AP_ERR_INAPP_CKSUM);
        endif

        if (auth_hdr.authenticator.cksum type is not supported) then
                error_out(KDC_ERR_SUMTYPE_NOSUPP);
        endif
        if (auth_hdr.authenticator.cksum is not both collision-proof and =
keyed) then
                error_out(KRB_AP_ERR_INAPP_CKSUM);
        endif

        set computed_checksum :=3D checksum(req);
        if (computed_checksum !=3D auth_hdr.authenticatory.cksum) then
                error_out(KRB_AP_ERR_MODIFIED);
        endif

        server :=3D lookup(req.sname,realm);

        if (!server) then
```

```
        if (is_foreign_tgt_name(req.sname)) then
                server :=3D best_intermediate_tgs(req.sname);
        else
                /* no server in Database */
                error_out(KDC_ERR_S_PRINCIPAL_UNKNOWN);
        endif
endif

session :=3D generate_random_session_key();

use_etype :=3D first supported etype in req.etypes;

if (no support for req.etypes) then
        error_out(KDC_ERR_ETYPE_NOSUPP);
endif

new_tkt.vno :=3D ticket version; /* =3D 5 */
new_tkt.sname :=3D req.sname;
new_tkt.srealm :=3D realm;
reset all flags in new_tkt.flags;

/* It should be noted that local policy may affect the  */
/* processing of any of these flags.  For example, some */
/* realms may refuse to issue renewable tickets         */


new_tkt.caddr :=3D tgt.caddr;
resp.caddr :=3D NULL; /* We only include this if they change */
if (req.kdc-options.FORWARDABLE is set) then
        if (tgt.flags.FORWARDABLE is reset) then
                error_out(KDC_ERR_BADOPTION);
        endif
        set new_tkt.flags.FORWARDABLE;
endif
if (req.kdc-options.FORWARDED is set) then
        if (tgt.flags.FORWARDABLE is reset) then
                error_out(KDC_ERR_BADOPTION);
        endif
        set new_tkt.flags.FORWARDED;
        new_tkt.caddr :=3D req.addresses;
        resp.caddr :=3D req.addresses;
endif
if (tgt.flags.FORWARDED is set) then
        set new_tkt.flags.FORWARDED;
endif

if (req.kdc-options.PROXIABLE is set) then
        if (tgt.flags.PROXIABLE is reset)
                error_out(KDC_ERR_BADOPTION);
        endif
        set new_tkt.flags.PROXIABLE;
endif
if (req.kdc-options.PROXY is set) then
        if (tgt.flags.PROXIABLE is reset) then
                error_out(KDC_ERR_BADOPTION);
        endif
        set new_tkt.flags.PROXY;
        new_tkt.caddr :=3D req.addresses;
        resp.caddr :=3D req.addresses;
endif

if (req.kdc-options.ALLOW-POSTDATE is set) then
        if (tgt.flags.MAY-POSTDATE is reset)
```

```
                        error_out(KDC_ERR_BADOPTION);
                endif
                set new_tkt.flags.MAY-POSTDATE;
        endif
        if (req.kdc-options.POSTDATED is set) then
                if (tgt.flags.MAY-POSTDATE is reset) then
                        error_out(KDC_ERR_BADOPTION);
                endif
                set new_tkt.flags.POSTDATED;
                set new_tkt.flags.INVALID;
                if (against_postdate_policy(req.from)) then
                        error_out(KDC_ERR_POLICY);
                endif
                new_tkt.starttime :=3D req.from;
        endif


        if (req.kdc-options.VALIDATE is set) then
                if (tgt.flags.INVALID is reset) then
                        error_out(KDC_ERR_POLICY);
                endif
                if (tgt.starttime > kdc_time) then
                        error_out(KRB_AP_ERR_NYV);
                endif
                if (check_hot_list(tgt)) then
                        error_out(KRB_AP_ERR_REPEAT);
                endif
                tkt :=3D tgt;
                reset new_tkt.flags.INVALID;
        endif

        if (req.kdc-options.(any flag except ENC-TKT-IN-SKEY, RENEW,
                             and those already processed) is set) then
                error_out(KDC_ERR_BADOPTION);
        endif

        new_tkt.authtime :=3D tgt.authtime;

        if (req.kdc-options.RENEW is set) then
          /* Note that if the endtime has already passed, the ticket =
would  */
          /* have been rejected in the initial authentication stage, so  =
   */
          /* there is no need to check again here                        =
   */
                if (tgt.flags.RENEWABLE is reset) then
                        error_out(KDC_ERR_BADOPTION);
                endif
                if (tgt.renew-till < kdc_time) then
                        error_out(KRB_AP_ERR_TKT_EXPIRED);
                endif
                tkt :=3D tgt;
                new_tkt.starttime :=3D kdc_time;
                old_life :=3D tgt.endttime - tgt.starttime;
                new_tkt.endtime :=3D min(tgt.renew-till,
                                        new_tkt.starttime + old_life);
        else
                new_tkt.starttime :=3D kdc_time;
                if (req.till =3D 0) then
                        till :=3D infinity;
                else
                        till :=3D req.till;
                endif
```

```
                new_tkt.endtime :=3D min(till,
                                        =
new_tkt.starttime+client.max_life,
                                        =
new_tkt.starttime+server.max_life,
                                        =
new_tkt.starttime+max_life_for_realm,
                                        tgt.endtime);



                if ((req.kdc-options.RENEWABLE-OK is set) and
                    (new_tkt.endtime < req.till) and
                    (tgt.flags.RENEWABLE is set) then
                        /* we set the RENEWABLE option for later =
processing */
                        set req.kdc-options.RENEWABLE;
                        req.rtime :=3D min(req.till, tgt.renew-till);
                endif
        endif

        if (req.rtime =3D 0) then
                rtime :=3D infinity;
        else
                rtime :=3D req.rtime;
        endif

        if ((req.kdc-options.RENEWABLE is set) and
            (tgt.flags.RENEWABLE is set)) then
                set new_tkt.flags.RENEWABLE;
                new_tkt.renew-till :=3D min(rtime,
                                           =
new_tkt.starttime+client.max_rlife,
                                           =
new_tkt.starttime+server.max_rlife,
                                           =
new_tkt.starttime+max_rlife_for_realm,
                                           tgt.renew-till);
        else
                new_tkt.renew-till :=3D OMIT; /* leave the renew-till =
field out */
        endif
        if (req.enc-authorization-data is present) then
                decrypt req.enc-authorization-data into =
decrypted_authorization_data
                        using auth_hdr.authenticator.subkey;
                if (decrypt_error()) then
                        error_out(KRB_AP_ERR_BAD_INTEGRITY);
                endif
        endif
        new_tkt.authorization_data :=3D =
req.auth_hdr.ticket.authorization_data +
                                  decrypted_authorization_data;

        new_tkt.key :=3D session;
        new_tkt.crealm :=3D tgt.crealm;
        new_tkt.cname :=3D req.auth_hdr.ticket.cname;

        if (realm_tgt_is_for(tgt) :=3D tgt.realm) then
                /* tgt issued by local realm */
                new_tkt.transited :=3D tgt.transited;
        else
                /* was issued for this realm by some other realm */
```

```
                if (tgt.transited.tr-type not supported) then
                        error_out(KDC_ERR_TRTYPE_NOSUPP);
                endif
                new_tkt.transited :=3D compress_transited(tgt.transited =
+ tgt.realm)



                /* Don't check transited field if TGT for foreign realm,=20
                 * or requested not to check */
                if (is_not_foreign_tgt_name(new_tkt.server)=20
                   && req.kdc-options.DISABLE-TRANSITED-CHECK not set) =
then
                        /* Check it, so end-server does not have to=20
                         * but don't fail, end-server may still accept =
it */
                        if (check_transited_field(new_tkt.transited) =
=3D=3D OK)
                              set =
new_tkt.flags.TRANSITED-POLICY-CHECKED;
                        endif
                endif
        endif

        encode encrypted part of new_tkt into OCTET STRING;
        if (req.kdc-options.ENC-TKT-IN-SKEY is set) then
                if (server not specified) then
                        server =3D req.second_ticket.client;
                endif
                if ((req.second_ticket is not a TGT) or
                    (req.second_ticket.client !=3D server)) then
                        error_out(KDC_ERR_POLICY);
                endif

                new_tkt.enc-part :=3D encrypt OCTET STRING using
                        using etype_for_key(second-ticket.key), =
second-ticket.key;
        else
                new_tkt.enc-part :=3D encrypt OCTET STRING
                        using etype_for_key(server.key), server.key, =
server.p_kvno;
        endif

        resp.pvno :=3D 5;
        resp.msg-type :=3D KRB_TGS_REP;
        resp.crealm :=3D tgt.crealm;
        resp.cname :=3D tgt.cname;
        resp.ticket :=3D new_tkt;

        resp.key :=3D session;
        resp.nonce :=3D req.nonce;
        resp.last-req :=3D fetch_last_request_info(client);
        resp.flags :=3D new_tkt.flags;

        resp.authtime :=3D new_tkt.authtime;
        resp.starttime :=3D new_tkt.starttime;
        resp.endtime :=3D new_tkt.endtime;

        omit resp.key-expiration;

        resp.sname :=3D new_tkt.sname;
        resp.realm :=3D new_tkt.realm;
```

```
        if (new_tkt.flags.RENEWABLE) then
                resp.renew-till :=3D new_tkt.renew-till;
        endif



        encode body of reply into OCTET STRING;

        if (req.padata.authenticator.subkey)
                resp.enc-part :=3D encrypt OCTET STRING using use_etype,
                        req.padata.authenticator.subkey;
        else resp.enc-part :=3D encrypt OCTET STRING using use_etype, =
tgt.key;

        send(resp);

=09
```

A.7. KRB_TGS_REP verification

```
        decode response into resp;

        if (resp.msg-type =3D KRB_ERROR) then
                process_error(resp);
                return;
        endif

        /* On error, discard the response, and zero the session key from
        the response immediately */

        if (req.padata.authenticator.subkey)
                unencrypted part of resp :=3D decode of decrypt of =
resp.enc-part
                        using resp.enc-part.etype and subkey;
        else unencrypted part of resp :=3D decode of decrypt of =
resp.enc-part
                                using resp.enc-part.etype and tgt's =
session key;
        if (common_as_rep_tgs_rep_checks fail) then
                destroy resp.key;
                return error;
        endif

        check authorization_data as necessary;
        save_for_later(ticket,session,client,server,times,flags);
```

A.8. Authenticator generation

```
        body.authenticator-vno :=3D authenticator vno; /* =3D 5 */
        body.cname, body.crealm :=3D client name;
        if (supplying checksum) then
                body.cksum :=3D checksum;
        endif
        get system_time;
        body.ctime, body.cusec :=3D system_time;
        if (selecting sub-session key) then
                select sub-session key;
                body.subkey :=3D sub-session key;
        endif
        if (using sequence numbers) then
                select initial sequence number;
                body.seq-number :=3D initial sequence;
```

```
        endif


A.9. KRB_AP_REQ generation

        obtain ticket and session_key from cache;

        packet.pvno :=3D protocol version; /* 5 */
        packet.msg-type :=3D message type; /* KRB_AP_REQ */

        if (desired(MUTUAL_AUTHENTICATION)) then
                set packet.ap-options.MUTUAL-REQUIRED;
        else
                reset packet.ap-options.MUTUAL-REQUIRED;
        endif
        if (using session key for ticket) then
                set packet.ap-options.USE-SESSION-KEY;
        else
                reset packet.ap-options.USE-SESSION-KEY;
        endif
        packet.ticket :=3D ticket; /* ticket */
        generate authenticator;
        encode authenticator into OCTET STRING;
        encrypt OCTET STRING into packet.authenticator using =
session_key;

A.10. KRB_AP_REQ verification

        receive packet;
        if (packet.pvno !=3D 5) then
                either process using other protocol spec
                or error_out(KRB_AP_ERR_BADVERSION);
        endif
        if (packet.msg-type !=3D KRB_AP_REQ) then
                error_out(KRB_AP_ERR_MSG_TYPE);
        endif
        if (packet.ticket.tkt_vno !=3D 5) then
                either process using other protocol spec
                or error_out(KRB_AP_ERR_BADVERSION);
        endif
        if (packet.ap_options.USE-SESSION-KEY is set) then
                retrieve session key from ticket-granting ticket for
                 packet.ticket.{sname,srealm,enc-part.etype};
        else
                retrieve service key for
                 =
packet.ticket.{sname,srealm,enc-part.etype,enc-part.skvno};
        endif
        if (no_key_available) then
                if (cannot_find_specified_skvno) then
                        error_out(KRB_AP_ERR_BADKEYVER);
                else
                        error_out(KRB_AP_ERR_NOKEY);
                endif
        endif
        decrypt packet.ticket.enc-part into decr_ticket using retrieved =
key;
        if (decryption_error()) then
                error_out(KRB_AP_ERR_BAD_INTEGRITY);
        endif


        decrypt packet.authenticator into decr_authenticator
```

```
                using decr_ticket.key;
        if (decryption_error()) then
                error_out(KRB_AP_ERR_BAD_INTEGRITY);
        endif
        if (decr_authenticator.{cname,crealm} !=3D
            decr_ticket.{cname,crealm}) then
                error_out(KRB_AP_ERR_BADMATCH);
        endif
        if (decr_ticket.caddr is present) then
                if (sender_address(packet) is not in decr_ticket.caddr) =
then
                        error_out(KRB_AP_ERR_BADADDR);
                endif
        elseif (application requires addresses) then
                error_out(KRB_AP_ERR_BADADDR);
        endif
        if (not in_clock_skew(decr_authenticator.ctime,
                              decr_authenticator.cusec)) then
                error_out(KRB_AP_ERR_SKEW);
        endif
        if (repeated(decr_authenticator.{ctime,cusec,cname,crealm})) =
then
                error_out(KRB_AP_ERR_REPEAT);
        endif
        save_identifier(decr_authenticator.{ctime,cusec,cname,crealm});
        get system_time;
        if ((decr_ticket.starttime-system_time > CLOCK_SKEW) or
            (decr_ticket.flags.INVALID is set)) then
                /* it hasn't yet become valid */
                error_out(KRB_AP_ERR_TKT_NYV);
        endif
        if (system_time-decr_ticket.endtime > CLOCK_SKEW) then
                error_out(KRB_AP_ERR_TKT_EXPIRED);
        endif
        if (decr_ticket.transited) then
            /* caller may ignore the TRANSITED-POLICY-CHECKED and do
             * check anyway */
            if (decr_ticket.flags.TRANSITED-POLICY-CHECKED not set) then
                 if (check_transited_field(decr_ticket.transited) then
                      error_out(KDC_AP_PATH_NOT_ACCPETED);
                 endif
            endif
        endif
        /* caller must check decr_ticket.flags for any pertinent details =
*/
        return(OK, decr_ticket, packet.ap_options.MUTUAL-REQUIRED);
```

A.11. KRB_AP_REP generation

```
        packet.pvno :=3D protocol version; /* 5 */
        packet.msg-type :=3D message type; /* KRB_AP_REP */

        body.ctime :=3D packet.ctime;
        body.cusec :=3D packet.cusec;
        if (selecting sub-session key) then
                select sub-session key;
                body.subkey :=3D sub-session key;
        endif

        if (using sequence numbers) then
                select initial sequence number;
                body.seq-number :=3D initial sequence;
        endif
```

```
        encode body into OCTET STRING;

        select encryption type;
        encrypt OCTET STRING into packet.enc-part;
```

A.12. KRB_AP_REP verification

```
        receive packet;
        if (packet.pvno !=3D 5) then
                either process using other protocol spec
                or error_out(KRB_AP_ERR_BADVERSION);
        endif
        if (packet.msg-type !=3D KRB_AP_REP) then
                error_out(KRB_AP_ERR_MSG_TYPE);
        endif
        cleartext :=3D decrypt(packet.enc-part) using ticket's session =
key;
        if (decryption_error()) then
                error_out(KRB_AP_ERR_BAD_INTEGRITY);
        endif
        if (cleartext.ctime !=3D authenticator.ctime) then
                error_out(KRB_AP_ERR_MUT_FAIL);
        endif
        if (cleartext.cusec !=3D authenticator.cusec) then
                error_out(KRB_AP_ERR_MUT_FAIL);
        endif
        if (cleartext.subkey is present) then
                save cleartext.subkey for future use;
        endif
        if (cleartext.seq-number is present) then
                save cleartext.seq-number for future verifications;
        endif
        return(AUTHENTICATION_SUCCEEDED);
```

A.13. KRB_SAFE generation

```
        collect user data in buffer;

        /* assemble packet: */
        packet.pvno :=3D protocol version; /* 5 */
        packet.msg-type :=3D message type; /* KRB_SAFE */

        body.user-data :=3D buffer; /* DATA */
        if (using timestamp) then
                get system_time;
                body.timestamp, body.usec :=3D system_time;
        endif
        if (using sequence numbers) then
                body.seq-number :=3D sequence number;
        endif
        body.s-address :=3D sender host addresses;
        if (only one recipient) then
                body.r-address :=3D recipient host address;
        endif


        checksum.cksumtype :=3D checksum type;
        compute checksum over body;
        checksum.checksum :=3D checksum value; /* checksum.checksum */
        packet.cksum :=3D checksum;
        packet.safe-body :=3D body;
```

```
A.14. KRB_SAFE verification

        receive packet;
        if (packet.pvno !=3D 5) then
                either process using other protocol spec
                or error_out(KRB_AP_ERR_BADVERSION);
        endif
        if (packet.msg-type !=3D KRB_SAFE) then
                error_out(KRB_AP_ERR_MSG_TYPE);
        endif
        if (packet.checksum.cksumtype is not both collision-proof and =
keyed) then
                error_out(KRB_AP_ERR_INAPP_CKSUM);
        endif
        if (safe_priv_common_checks_ok(packet)) then
                set computed_checksum :=3D checksum(packet.body);
                if (computed_checksum !=3D packet.checksum) then
                        error_out(KRB_AP_ERR_MODIFIED);
                endif
                return (packet, PACKET_IS_GENUINE);
        else
                return common_checks_error;
        endif

A.15. KRB_SAFE and KRB_PRIV common checks

        if (packet.s-address !=3D O/S_sender(packet)) then
                /* O/S report of sender not who claims to have sent it =
*/
                error_out(KRB_AP_ERR_BADADDR);
        endif
        if ((packet.r-address is present) and
            (packet.r-address !=3D local_host_address)) then
                /* was not sent to proper place */
                error_out(KRB_AP_ERR_BADADDR);
        endif
        if (((packet.timestamp is present) and
             (not in_clock_skew(packet.timestamp,packet.usec))) or
            (packet.timestamp is not present and timestamp expected)) =
then
                error_out(KRB_AP_ERR_SKEW);
        endif
        if (repeated(packet.timestamp,packet.usec,packet.s-address)) =
then
                error_out(KRB_AP_ERR_REPEAT);
        endif

        if (((packet.seq-number is present) and
             ((not in_sequence(packet.seq-number)))) or
            (packet.seq-number is not present and sequence expected)) =
then
                error_out(KRB_AP_ERR_BADORDER);
        endif



        if (packet.timestamp not present and packet.seq-number not =
present) then
                error_out(KRB_AP_ERR_MODIFIED);
        endif

        save_identifier(packet.{timestamp,usec,s-address},
                        sender_principal(packet));
```

```
        return PACKET_IS_OK;
```

A.16. KRB_PRIV generation

```
        collect user data in buffer;

        /* assemble packet: */
        packet.pvno :=3D protocol version; /* 5 */
        packet.msg-type :=3D message type; /* KRB_PRIV */

        packet.enc-part.etype :=3D encryption type;

        body.user-data :=3D buffer;
        if (using timestamp) then
                get system_time;
                body.timestamp, body.usec :=3D system_time;
        endif
        if (using sequence numbers) then
                body.seq-number :=3D sequence number;
        endif
        body.s-address :=3D sender host addresses;
        if (only one recipient) then
                body.r-address :=3D recipient host address;
        endif

        encode body into OCTET STRING;

        select encryption type;
        encrypt OCTET STRING into packet.enc-part.cipher;
```

A.17. KRB_PRIV verification

```
        receive packet;
        if (packet.pvno !=3D 5) then
                either process using other protocol spec
                or error_out(KRB_AP_ERR_BADVERSION);
        endif
        if (packet.msg-type !=3D KRB_PRIV) then
                error_out(KRB_AP_ERR_MSG_TYPE);
        endif

        cleartext :=3D decrypt(packet.enc-part) using negotiated key;
        if (decryption_error()) then
                error_out(KRB_AP_ERR_BAD_INTEGRITY);
        endif

        if (safe_priv_common_checks_ok(cleartext)) then
                return(cleartext.DATA, =
PACKET_IS_GENUINE_AND_UNMODIFIED);
        else
                return common_checks_error;
        endif
```

A.18. KRB_CRED generation

```
        invoke KRB_TGS; /* obtain tickets to be provided to peer */

        /* assemble packet: */
        packet.pvno :=3D protocol version; /* 5 */
        packet.msg-type :=3D message type; /* KRB_CRED */
```

```
        for (tickets[n] in tickets to be forwarded) do
                packet.tickets[n] =3D tickets[n].ticket;
        done

        packet.enc-part.etype :=3D encryption type;

        for (ticket[n] in tickets to be forwarded) do
                body.ticket-info[n].key =3D tickets[n].session;
                body.ticket-info[n].prealm =3D tickets[n].crealm;
                body.ticket-info[n].pname =3D tickets[n].cname;
                body.ticket-info[n].flags =3D tickets[n].flags;
                body.ticket-info[n].authtime =3D tickets[n].authtime;
                body.ticket-info[n].starttime =3D tickets[n].starttime;
                body.ticket-info[n].endtime =3D tickets[n].endtime;
                body.ticket-info[n].renew-till =3D =
tickets[n].renew-till;
                body.ticket-info[n].srealm =3D tickets[n].srealm;
                body.ticket-info[n].sname =3D tickets[n].sname;
                body.ticket-info[n].caddr =3D tickets[n].caddr;
        done

        get system_time;
        body.timestamp, body.usec :=3D system_time;

        if (using nonce) then
                body.nonce :=3D nonce;
        endif

        if (using s-address) then
                body.s-address :=3D sender host addresses;
        endif
        if (limited recipients) then
                body.r-address :=3D recipient host address;
        endif

        encode body into OCTET STRING;

        select encryption type;
        encrypt OCTET STRING into packet.enc-part.cipher
               using negotiated encryption key;
```

A.19. KRB_CRED verification

```
        receive packet;
        if (packet.pvno !=3D 5) then
                either process using other protocol spec
                or error_out(KRB_AP_ERR_BADVERSION);
        endif
        if (packet.msg-type !=3D KRB_CRED) then
                error_out(KRB_AP_ERR_MSG_TYPE);
        endif

        cleartext :=3D decrypt(packet.enc-part) using negotiated key;
        if (decryption_error()) then
                error_out(KRB_AP_ERR_BAD_INTEGRITY);
        endif
        if ((packet.r-address is present or required) and
           (packet.s-address !=3D O/S_sender(packet)) then
                /* O/S report of sender not who claims to have sent it =
*/
                error_out(KRB_AP_ERR_BADADDR);
        endif
        if ((packet.r-address is present) and
```

```
            (packet.r-address !=3D local_host_address)) then
                /* was not sent to proper place */
                error_out(KRB_AP_ERR_BADADDR);
        endif
        if (not in_clock_skew(packet.timestamp,packet.usec)) then
                error_out(KRB_AP_ERR_SKEW);
        endif
        if (repeated(packet.timestamp,packet.usec,packet.s-address)) =
then
                error_out(KRB_AP_ERR_REPEAT);
        endif
        if (packet.nonce is required or present) and
           (packet.nonce !=3D expected-nonce) then
                error_out(KRB_AP_ERR_MODIFIED);
        endif

        for (ticket[n] in tickets that were forwarded) do
                save_for_later(ticket[n],key[n],principal[n],
                                server[n],times[n],flags[n]);
        return
```

A.20. KRB_ERROR generation

```
        /* assemble packet: */
        packet.pvno :=3D protocol version; /* 5 */
        packet.msg-type :=3D message type; /* KRB_ERROR */

        get system_time;
        packet.stime, packet.susec :=3D system_time;
        packet.realm, packet.sname :=3D server name;

        if (client time available) then
                packet.ctime, packet.cusec :=3D client_time;
        endif



        packet.error-code :=3D error code;
        if (client name available) then
                packet.cname, packet.crealm :=3D client name;
        endif
        if (error text available) then
                packet.e-text :=3D error text;
        endif
        if (error data available) then
                packet.e-data :=3D error data;
        endif
```

B. Definition of common authorization data elements

This appendix contains the definitions of common authorization data elements. These common authorization data elements are recursivly defined, meaning the ad-data for these types will itself contain a sequence of authorization data whose interpretation is affected by the encapsulating element. Depending on the meaning of the encapsulating element, the encapsulated elements may be ignored, might be interpreted as issued directly by the KDC, or they might be stored in a separate plaintext part of the ticket. The types of the encapsulating elements are specified as part of the Kerberos specification because the behavior based on these values should be understood across implementations whereas other elements need only be understood by the applications which they affect.

In the definitions that follow, the value of the ad-type for the element will be specified in the subsection number, and the value of the ad-data will be as shown in the ASN.1 structure that follows the subsection heading.

B.1. If relevant

```
AD-IF-RELEVANT   AuthorizationData
```

AD elements encapsulated within the if-relevant element are intended for interpretation only by application servers that understand the particular ad-type of the embedded element. Application servers that do not understand the type of an element embedded within the if-relevant element may ignore the uninterpretable element. This element promotes interoperability across implementations which may have local extensions for authorization.

B.2. Intended for server

```
AD-INTENDED-FOR-SERVER   SEQUENCE {
         intended-server[0]     SEQUENCE OF PrincipalName
         elements[1]            AuthorizationData
}
```

AD elements encapsulated within the intended-for-server element may be ignored if the application server is not in the list of principal names of intended servers. Further, a KDC issuing a ticket for an application server can remove this element if the application server is not in the list of intended servers.

Application servers should check for their principal name in the intended-server field of this element. If their principal name is not found, this element should be ignored. If found, then the encapsulated elements should be evaluated in the same manner as if they were present in the top level authorization data field. Applications and application servers that do not implement this element should reject tickets that contain authorization data elements of this type.

B.3. Intended for application class

AD-INTENDED-FOR-APPLICATION-CLASS SEQUENCE { intended-application-class[0] SEQUENCE OF GeneralString elements[1] AuthorizationData } AD elements encapsulated within the intended-for-application-class element may be ignored if the application server is not in one of the named classes of application servers. Examples of application server classes include "FILESYSTEM", and other kinds of servers.=20
This element and the elements it encapsulates may be safely ignored by applications, application servers, and KDCs that do not implement this element.

B.4. KDC Issued

```
AD-KDCIssued   SEQUENCE {
               ad-checksum[0]    Checksum,
                i-realm[1]       Realm OPTIONAL,
                i-sname[2]       PrincipalName OPTIONAL,
               elements[3]       AuthorizationData.
}
```

ad-checksum
A checksum over the elements field using a cryptographic checksum method that is identical to the checksum used to protect the ticket itself (i.e. using the same hash function and the same encryption algorithm used to encrypt the ticket) and using a key derived from the

same key used to protect the ticket.

i-realm, i-sname
: The name of the issuing principal if different from the KDC itself. This field would be used when the KDC can verify the authenticity of elements signed by the issuing principal and it allows this KDC to notify the application server of the validity of those elements.

elements
: A sequence of authorization data elements issued by the KDC.

The KDC-issued ad-data field is intended to provide a means for Kerberos principal credentials to embed within themselves privilege attributes and other mechanisms for positive authorization, amplifying the privileges of the principal beyond what can be done using a credentials without such an a-data element.

This can not be provided without this element because the definition of the authorization-data field allows elements to be added at will by the bearer of a TGT at the time that they request service tickets and elements may also be added to a delegated ticket by inclusion in the authenticator.

For KDC-issued elements this is prevented because the elements are signed by the KDC by including a checksum encrypted using the server's key (the same key used to encrypt the ticket - or a key derived from that key). Elements encapsulated with in the KDC-issued element will be ignored by the application server if this "signature" is not present. Further, elements encapsulated within this element from a ticket granting ticket may be interpreted by the KDC, and used as a basis according to policy for including new signed elements within derivative tickets, but they will not be copied to a derivative ticket directly. If they are copied directly to a derivative ticket by a KDC that is not aware of this element, the signature will not be correct for the application ticket elements, and the field will be ignored by the application server.

This element and the elements it encapsulates may be safely ignored by applications, application servers, and KDCs that do not implement this element.

B.5. And-Or

```
AD-AND-OR           SEQUENCE {
                        condition-count[0]    INTEGER,
                        elements[1]           AuthorizationData
}=20
```

When restrictive AD elements encapsulated within the and-or element are encountered, only the number specified in condition-count of the encapsulated conditions must be met in order to satisfy this element. This element may be used to implement an "or" operation by setting the condition-count field to 1, and it may specify an "and" operation by setting the condition count to the number of embedded elements. Application servers that do not implement this element must reject tickets that contain authorization data elements of this type.

B.6. Mandatory ticket extensions

```
AD-Mandatory-Ticket-Extensions          SEQUENCE {
                        te-type[0]      INTEGER,
                        te-checksum[0]   Checksum
}=20
```

An authorization data element of type mandatory-ticket-extensions specifies the type and a collision-proof checksum using the same hash algorithm used

to protect the integrity of the ticket itself. This checksum will be calculated over an individual extension field of the type indicated. If there are more than one extension, multiple Mandatory-Ticket-Extensions authorization data elements may be present, each with a checksum for a different extension field. This restriction indicates that the ticket should not be accepted if a ticket extension is not present in the ticket for which the type and checksum do not match that checksum specified in the authorization data element. Note that although the type is redundant for the purposes of the comparison, it makes the comparison easier when multiple extensions are present. Application servers that do not implement this element must reject tickets that contain authorization data elements of this type.

B.7. Authorization Data in ticket extensions

AD-IN-Ticket-Extensions Checksum

An authorization data element of type in-ticket-extensions specifies a collision-proof checksum using the same hash algorithm used to protect the integrity of the ticket itself. This checksum is calculated over a separate external AuthorizationData field carried in the ticket extensions. Application servers that do not implement this element must reject tickets that contain authorization data elements of this type. Application servers that do implement this element will search the ticket extensions for authorization data fields, calculate the specified checksum over each authorization data field and look for one matching the checksum in this in-ticket-extensions element. If not found, then the ticket must be rejected. If found, the corresponding authorization data elements will be interpreted in the same manner as if they were contained in the top level authorization data field.

Note that if multiple external authorization data fields are present in a ticket, each will have a corresponding element of type in-ticket-extensions in the top level authorization data field, and the external entries will be linked to the corresponding element by their checksums.

C. Definition of common ticket extensions

This appendix contains the definitions of common ticket extensions. Support for these extensions is optional. However, certain extensions have associated authorization data elements that may require rejection of a ticket containing an extension by application servers that do not implement the particular extension. Other extensions have been defined beyond those described in this specification. Such extensions are described elsewhere and for some of those extensions the reserved number may be found in the list of constants.
It is known that older versions of Kerberos did not support this field, and that some clients will strip this field from a ticket when they parse and then reassemble a ticket as it is passed to the application servers. The presence of the extension will not break such clients, but any functionally dependent on the extensions will not work when such tickets are handled by old clients. In such situations, some implementation may use alternate methods to transmit the information in the extensions field.

C.1. Null ticket extension

TE-NullExtension OctetString -- The empty Octet String

The te-data field in the null ticket extension is an octet string of length zero. This extension may be included in a ticket granting ticket so that the KDC can determine on presentation of the ticket granting ticket whether the

client software will strip the extensions field. =20

C.2. External Authorization Data

TE-ExternalAuthorizationData AuthorizationData

The te-data field in the external authorization data ticket extension is field of type AuthorizationData containing one or more authorization data elements. If present, a corresponding authorization data element will be present in the primary authorization data for the ticket and that element will contain a checksum of the external authorization data ticket extension.

D. Significant changes since RFC 1510

Commentary

Section 1: The preamble and introduction does not define the protocol, mention is made in the introduction regarding the ability to rely on the KDC to check the transited field, and on the inclusion of a flag in a ticket indicating that this check has occurred. This is a new capability not present in RFC1510. Pre-existing implementation may ignore or not set this flag without negative security implications.

The definition of the secret key says that in the case of a user the key may be derived from a password. In 1510, it said that the key was derived from the password. This change was made to accommodate situations where the user key might be stored on a smart-card, or otherwise obtained independent of a password.

The introduction also mentions the use of public key for initial authentication in Kerberos by reference. RFC1510 did not include such a reference.

Section 1.2 was added to explain that while Kerberos provides authentication of a named principal, it is still the responsibility of the application to ensure that the authenticated name is the entity with which the application wishes to communicate. Because section 1.2 is completely new, I am particularly interested in suggestions to improve the wording of this section. Sections 1.2-4 were renumbered.

Section 2: No changes were made to existing options and flags specified in RFC1510, though some of the sections in the specification were renumbered, and text was revised to make the description and intent of existing options clearer, especially with respect to the ENC-TKT-IN-SKEY option (now section 2.9.3) which is used for user-to-user authentication. New options and ticket flags added since RFC1510 include transited policy checking (section 2.7), anonymous tickets (section 2.8) and name canonicalization (section 2.9.1).

Section 3: Added mention of the optional checksum field in the KRB-ERROR message. Added mention of name canonicalization and anonymous tickets in exposition on KDC options. Mention of the name canonicalization case is included in the description of the KDC reply (3.1.3). A warning regarding generation of session keys for application use was added, urging the inclusion of key entropy from the KDC generated session key in the ticket. An example regarding use of the subsession key was added to section 3.2.6. Descriptions of the pa-etype-info, and pa-pw-salt preauthentication data items were added.

Changes to section 4: Added language about who has access to the keys in the Kerberos database. Also made it clear that KDC's may obtain the information from some database field through other means - for example, one form of pkinit may extract some of these fields from a certificate.

Regarding the discussion on the list regarding the use of tamper resistant hardware to store keys, I was not able to determine specific suggested changes to the text in the RFC regarding this. Much of this discussion centers around particular implementations. I did however loosen the wording about the database so as not to preclude keys that can not be extracted in the clear from such hardware.

Section 5: A statement regarding the carrying of unrecognized additional fields in ASN.1 encoding through in tickets was added (still waiting on some better text regarding this).

Ticket flags and KDC options were added to support the new functions described elsewhere in this document. The encoding of the options flags are now described to be no less than 32 bits, and the smallest number of bits beyond 32 needed to encode any set bits. It also describes the encoding of the bitstring as using "unnamed" bits.

An optional ticket extensions field was added to support the carrying of auxiliary data that allows the passing of auxiliary that is to accompany a ticket to the verifier.

(I would like to drop the part about optionally appending it of the opaque part of the ciphertext. We are still waiting on some text regarding how to assure backward compatibility).

(Still pending, Tom Yu's request to change the application codes on KDC message to indicate which minor rev of the protocol - I think this might break things, but am not sure).

Definition of the PA-USE-SPECIFIED-KVNO preauthentication data field was added.

The optional e-cksum field was added to the KRB-ERROR message and the e-data filed was generalized for use in other than the KDC_ERR_PREAUTH_REQUIRED error. The TypedData structure was defined. Type tags for TypedData are defined in the same sequence as the PA-DATA type space to avoid confusion with the use of the PA-DATA namespace previously used for the e-data field for the KDC_ERR_PREAUTH_REQUIRED error.

Section 7: Words were added describing the convention that domain based realm names for newly created realms should be specified as upper case. This recommendation does not make lower case realm names illegal. Words were added highlighting that the slash separated components in the X500 style of realm names is consistent with existing RFC1510 based implementations, but that it conflicts with the general recommendation of X.500 name representation specified in RFC2253.

There were suggestions on the list regarding extensions to or new name types. These require discussion at the IETF meeting. My own feeling at this point is that in the absence of a strong consensus for adding new types at this time, I would rather not add new name types in the current draft, but leave things open for additions later.

Section 8: Since RFC1510, the definition of the TCP transport for Kerberos messages was added.

Section 9: Requirements for supporting DES3-CBC-SHA1-KD encryption and HMAC-SHA1-DES3-KD checksums were added.

I would like to make support for Rijndael mandatory and for us to have a SINGLE standard for use of Rijndale in these revisions.

Discussion

Section 8: Regarding the suggestion of weakening the requirement for use of port 88 for cases where the port can be looked up elsewhere - I did not incorporate this suggestion because cross realm authentication requires the ability to contact the appropriate KDC, and unless ALL implementations of Kerberos include support for finding such alternate port numbers, use of such KDC's would be non-interoperable.

[TM] Project Athena, Athena, and Kerberos are trademarks of the Massachusetts Institute of Technology (MIT). No commercial use of these trademarks may be made without prior written permission of MIT.

[1.1] Note, however, that many applications use Kerberos' functions only upon the initiation of a stream-based network connection. Unless an application subsequently provides integrity protection for the data stream, the identity verification applies only to the initiation of the connection, and does not guarantee that subsequent messages on the connection originate from the same principal.

[1.2] Secret and private are often used interchangeably in the literature. In our usage, it takes two (or more) to share a secret, thus a shared DES key is a secret key. Something is only private when no one but its owner knows it. Thus, in public key cryptosystems, one has a public and a private key.

[1.3] Of course, with appropriate permission the client could arrange registration of a separately-named principal in a remote realm, and engage in normal exchanges with that realm's services. However, for even small numbers of clients this becomes cumbersome, and more automatic methods as described here are necessary.

[2.1] Though it is permissible to request or issue tickets with no network addresses specified.

[2.2] It is important that the KDC be sent the name as typed by the user, and not only the canonical form of the name. If the domain name system was used to find the canonical name on the client side, the mapping is vulnerable. [3.1] The password-changing request must not be honored unless the requester can provide the old password (the user's current secret key). Otherwise, it would be possible for someone to walk up to an unattended session and change another user's password.

[3.2] To authenticate a user logging on to a local system, the credentials obtained in the AS exchange may first be used in a TGS exchange to obtain credentials for a local server. Those credentials must then be verified by a local server through successful completion of the Client/Server exchange.

[3.3] "Random" means that, among other things, it should be impossible to guess the next session key based on knowledge of past session keys. This can only be achieved in a pseudo-random number generator if it is based on cryptographic principles. It is more desirable to use a truly random number generator, such as one based on measurements of random physical phenomena.

[3.4] Tickets contain both an encrypted and unencrypted portion, so cleartext here refers to the entire unit, which can be copied from one message and replayed in another without any cryptographic skill.

[3.5] Note that this can make applications based on unreliable transports

difficult to code correctly. If the transport might deliver duplicated messages, either a new authenticator must be generated for each retry, or the application server must match requests and replies and replay the first reply in response to a detected duplicate.

[3.6] This allows easy implementation of user-to-user authentication [8], which uses ticket-granting ticket session keys in lieu of secret server keys in situations where such secret keys could be easily compromised.

[3.7]Note also that the rejection here is restricted to authenticators from the same principal to the same server. Other client principals communicating with the same server principal should not be have their authenticators rejected if the time and microsecond fields happen to match some other client's authenticator.

[3.8] If this is not done, an attacker could subvert the authentication by recording the ticket and authenticator sent over the network to a server and replaying them following an event that caused the server to lose track of recently seen authenticators.

[3.9] In the Kerberos version 4 protocol, the timestamp in the reply was the client's timestamp plus one. This is not necessary in version 5 because version 5 messages are formatted in such a way that it is not possible to create the reply by judicious message surgery (even in encrypted form) without knowledge of the appropriate encryption keys.

[3.10] Note that for encrypting the KRB_AP_REP message, the sub-session key is not used, even if present in the Authenticator.

[3.11] Implementations of the protocol may wish to provide routines to choose subkeys based on session keys and random numbers and to generate a negotiated key to be returned in the KRB_AP_REP message.

[3.12]This can be accomplished in several ways. It might be known beforehand (since the realm is part of the principal identifier), it might be stored in a nameserver, or it might be obtained from a configuration file. If the realm to be used is obtained from a nameserver, there is a danger of being spoofed if the nameservice providing the realm name is not authenticated. This might result in the use of a realm which has been compromised, and would result in an attacker's ability to compromise the authentication of the application server to the client.

[3.13] If the client selects a sub-session key, care must be taken to ensure the randomness of the selected sub-session key. One approach would be to generate a random number and XOR it with the session key from the ticket-granting ticket.

[4.1] The implementation of the Kerberos server need not combine the database and the server on the same machine; it is feasible to store the principal database in, say, a network name service, as long as the entries stored therein are protected from disclosure to and modification by unauthorized parties. However, we recommend against such strategies, as they can make system management and threat analysis quite complex.

[4.2] See the discussion of the padata field in section 5.4.2 for details on why this can be useful.

Appendix C. PKINIT Specification

The PKINIT specification is currently still an IETF draft. This document complies only with the version of the PKINIT draft that is included in this section. The PacketCable security team will continue to track the progress of the PKINIT draft through the IETF. Note that the details of the first and second Oakley groups are provided in Appendix H of this specification.

INTERNET-DRAFT
draft-ietf-cat-kerberos-pk-init-16.txt
Updates: RFC 1510bis
expires June 25, 2002

Brian Tung
Clifford Neuman
USC/ISI
Matthew Hur
Cisco
Ari Medvinsky
Keen.com, Inc.
Sasha Medvinsky
Motorola
John Wray
Iris Associates, Inc.
Jonathan Trostle
Cisco

Public Key Cryptography for Initial Authentication in Kerberos

0. Status Of This Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC 2026. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.ietf.org (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

The distribution of this memo is unlimited. It is filed as draft-ietf-cat-kerberos-pk-init-16.txt, and expires June 25, 2002. Please send comments to the authors.

1. Abstract

This document defines extensions (PKINIT) to the Kerberos protocol specification (RFC 1510bis [1]) to provide a method for using public key cryptography during initial authentication. The methods defined specify the ways in which preauthentication data fields and error data fields in Kerberos messages are to be used to transport public key data.

2. Introduction

The popularity of public key cryptography has produced a desire for its support in Kerberos [2]. The advantages provided by public key cryptography include simplified key management (from the Kerberos perspective) and the ability to leverage existing and developing public key certification infrastructures.

Public key cryptography can be integrated into Kerberos in a number of ways. One is to associate a key pair with each realm, which can then be used to facilitate cross-realm authentication; this is the topic of another draft proposal. Another way is to allow users with public key certificates to use them in initial authentication. This is the concern of the current document.

PKINIT utilizes ephemeral-ephemeral Diffie-Hellman keys in combination with RSA keys as the primary, required mechanism. Note that PKINIT supports the use of separate signature and encryption keys.

PKINIT enables access to Kerberos-secured services based on initial authentication utilizing public key cryptography. PKINIT utilizes standard public key signature and encryption data formats within the standard Kerberos messages. The basic mechanism is as follows: The user sends an AS-REQ message to the KDC as before, except that if that user is to use public key cryptography in the initial authentication step, his certificate and a signature accompany the initial request in the preauthentication fields. Upon receipt of this request, the KDC verifies the certificate and issues a ticket granting ticket (TGT) as before, except that the encPart from the AS-REP message carrying the TGT is now encrypted utilizing either a Diffie-Hellman derived key or the user's public key. This message is authenticated utilizing the public key signature of the KDC.

Note that PKINIT does not require the use of certificates. A KDC may store the public key of a principal as part of that principal's record. In this scenario, the KDC is the trusted party that vouches for the principal (as in a standard, non-cross realm, Kerberos environment). Thus, for any principal, the KDC may maintain a symmetric key, a public key, or both.

The PKINIT specification may also be used as a building block for other specifications. PKINIT may be utilized to establish inter-realm keys for the purposes of issuing cross-realm service tickets. It may also be used to issue anonymous Kerberos tickets using the Diffie-Hellman option. Efforts are under way to draft specifications for these two application protocols.

Additionally, the PKINIT specification may be used for direct peer to peer authentication without contacting a central KDC. This application of PKINIT is based on concepts introduced in [6, 7]. For direct client-to-server authentication, the client uses PKINIT to authenticate to the end server (instead of a central KDC), which then issues a ticket for itself. This approach has an advantage over TLS [5] in that the server does not need to save state (cache session keys). Furthermore, an additional benefit is that Kerberos tickets can facilitate delegation (see [6]).

3. Proposed Extensions

This section describes extensions to RFC 1510bis for supporting the use of public key cryptography in the initial request for a ticket granting ticket (TGT).

In summary, the following change to RFC 1510bis is proposed:

* Users may authenticate using either a public key pair or a conventional (symmetric) key. If public key cryptography is used, public key data is transported in preauthentication data fields to help establish identity. The user presents a public key certificate and obtains an ordinary TGT that may be used for subsequent authentication, with such authentication using only conventional cryptography.

Section 3.1 provides definitions to help specify message formats. Section 3.2 describes the extensions for the initial authentication method.

3.1. Definitions

The extensions involve new preauthentication fields; we introduce the following preauthentication types:

PA-PK-AS-REQ	14
PA-PK-AS-REP	15

The extensions also involve new error types; we introduce the following types:

KDC_ERR_CLIENT_NOT_TRUSTED	62
KDC_ERR_KDC_NOT_TRUSTED	63
KDC_ERR_INVALID_SIG	64
KDC_ERR_KEY_TOO_WEAK	65
KDC_ERR_CERTIFICATE_MISMATCH	66
KDC_ERR_CANT_VERIFY_CERTIFICATE	70
KDC_ERR_INVALID_CERTIFICATE	71
KDC_ERR_REVOKED_CERTIFICATE	72
KDC_ERR_REVOCATION_STATUS_UNKNOWN	73
KDC_ERR_REVOCATION_STATUS_UNAVAILABLE	74
KDC_ERR_CLIENT_NAME_MISMATCH	75
KDC_ERR_KDC_NAME_MISMATCH	76

We utilize the following typed data for errors:

TD-PKINIT-CMS-CERTIFICATES	101
TD-KRB-PRINCIPAL	102
TD-KRB-REALM	103
TD-TRUSTED-CERTIFIERS	104
TD-CERTIFICATE-INDEX	105

We utilize the following encryption types (which map directly to OIDs):

dsaWithSHA1-CmsOID	9
md5WithRSAEncryption-CmsOID	10
sha1WithRSAEncryption-CmsOID	11
rc2CBC-EnvOID	12
rsaEncryption-EnvOID (PKCS#1 v1.5)	13
rsaES-OAEP-ENV-OID (PKCS#1 v2.0)	14
des-ede3-cbc-Env-OID	15

These mappings are provided so that a client may send the appropriate enctypes in the AS-REQ message in order to indicate support for the corresponding OIDs (for performing PKINIT). The above encryption types are utilized only within CMS structures within the PKINIT preauthentication fields. Their use within

the Kerberos EncryptedData structure is unspecified.

In many cases, PKINIT requires the encoding of the X.500 name of a certificate authority as a Realm. When such a name appears as a realm it will be represented using the "Other" form of the realm name as specified in the naming constraints section of RFC 1510bis. For a realm derived from an X.500 name, NAMETYPE will have the value X500-RFC2253. The full realm name will appear as follows:

<nametype> + ":" + <string>

where nametype is "X500-RFC2253" and string is the result of doing an RFC2253 encoding of the distinguished name, i.e.

"X500-RFC2253:" + RFC2253Encode(DistinguishedName)

where DistinguishedName is an X.500 name, and RFC2253Encode is a function returning a readable UTF encoding of an X.500 name, as defined by RFC 2253 [11] (part of LDAPv3 [15]).

Each component of a DistinguishedName is called a RelativeDistinguishedName, where a RelativeDistinguishedName is a SET OF AttributeTypeAndValue. RFC 2253 does not specify the order in which to encode the elements of the RelativeDistinguishedName and so to ensure that this encoding is unique, we add the following rule to those specified by RFC 2253:

When converting a multi-valued RelativeDistinguishedName to a string, the output consists of the string encodings of each AttributeTypeAndValue, in the same order as specified by the DER encoding.

Similarly, in cases where the KDC does not provide a specific policy-based mapping from the X.500 name or X.509 Version 3 SubjectAltName extension in the user's certificate to a Kerberos principal name, PKINIT requires the direct encoding of the X.500 name as a PrincipalName. In this case, the name-type of the principal name MUST be set to KRB_NT-X500-PRINCIPAL. This new name type is defined in RFC 1510bis as:

KRB_NT_X500_PRINCIPAL 6

For this type, the name-string MUST be set as follows:

RFC2253Encode(DistinguishedName)

as described above. When this name type is used, the principal's realm MUST be set to the certificate authority's distinguished name using the X500-RFC2253 realm name format described earlier in this section.

Note that the same string may be represented using several different ASN.1 data types. As the result, the reverse conversion from an RFC2253-encoded principal name back to an X.500 name may not be unique and may result in an X.500 name that is not the same as the original X.500 name found in the client certificate.

RFC 1510bis describes an alternate encoding of an X.500 name into a realm name. However, as described in RFC 1510bis, the alternate encoding does not guarantee a unique mapping from a DistinguishedName inside a certificate into a realm name and similarly cannot be used to produce a unique principal name. PKINIT therefore uses an RFC 2253-based name mapping approach, as specified

above.

RFC 1510bis specifies the ASN.1 structure for PrincipalName as follows:

```
PrincipalName ::=   SEQUENCE {
                    name-type[0]     INTEGER,
                    name-string[1]   SEQUENCE OF GeneralString
}
```

The following rules relate to the matching of PrincipalNames with regard to the PKI name constraints for CAs as laid out in RFC 2459 [12]. In order to be regarded as a match (for permitted and excluded name trees), the following MUST be satisfied.

1. If the constraint is given as a user plus realm name, or as a client principal name plus realm name (as specified in RFC 1510bis), the realm name MUST be valid (see 2.a-d below) and the match MUST be exact, byte for byte.

2. If the constraint is given only as a realm name, matching depends on the type of the realm:

   a. If the realm contains a colon (':') before any equal sign ('='), it is treated as a realm of type Other, and MUST match exactly, byte for byte.

   b. Otherwise, if the realm name conforms to rules regarding the format of DNS names, it is considered a realm name of type Domain. The constraint may be given as a realm name 'FOO.BAR', which matches any PrincipalName within the realm 'FOO.BAR' but not those in subrealms such as 'CAR.FOO.BAR'. A constraint of the form '.FOO.BAR' matches PrincipalNames in subrealms of the form 'CAR.FOO.BAR' but not the realm 'FOO.BAR' itself.

   c. Otherwise, the realm name is invalid and does not match under any conditions.

3.1.1. Encryption and Key Formats

In the exposition below, we use the terms public key and private key generically. It should be understood that the term "public key" may be used to refer to either a public encryption key or a signature verification key, and that the term "private key" may be used to refer to either a private decryption key or a signature generation key. The fact that these are logically distinct does not preclude the assignment of bitwise identical keys for RSA keys.

In the case of Diffie-Hellman, the key is produced from the agreed bit string as follows:

* Truncate the bit string to the required length.
* Apply the specific cryptosystem's random-to-key function.

Appropriate key constraints for each valid cryptosystem are given in RFC 1510bis.

3.1.2. Algorithm Identifiers

PKINIT does not define, but does permit, the algorithm identifiers listed below.

3.1.2.1. Signature Algorithm Identifiers

The following signature algorithm identifiers specified in [8] and in [12] are used with PKINIT:

```
sha-1WithRSAEncryption (RSA with SHA1)
md5WithRSAEncryption   (RSA with MD5)
id-dsa-with-sha1       (DSA with SHA1)
```

3.1.2.2 Diffie-Hellman Key Agreement Algorithm Identifier

The following algorithm identifier shall be used within the SubjectPublicKeyInfo data structure: dhpublicnumber

This identifier and the associated algorithm parameters are specified in RFC 2459 [12].

3.1.2.3. Algorithm Identifiers for RSA Encryption

These algorithm identifiers are used inside the EnvelopedData data structure, for encrypting the temporary key with a public key:

```
rsaEncryption (RSA encryption, PKCS#1 v1.5)
id-RSAES-OAEP (RSA encryption, PKCS#1 v2.0)
```

Both of the above RSA encryption schemes are specified in [13]. Currently, only PKCS#1 v1.5 is specified by CMS [8], although the CMS specification says that it will likely include PKCS#1 v2.0 in the future. (PKCS#1 v2.0 addresses adaptive chosen ciphertext vulnerability discovered in PKCS#1 v1.5.)

3.1.2.4. Algorithm Identifiers for Encryption with Secret Keys

These algorithm identifiers are used inside the EnvelopedData data structure in the PKINIT Reply, for encrypting the reply key with the temporary key:

```
des-ede3-cbc (3-key 3-DES, CBC mode)
rc2-cbc      (RC2, CBC mode)
```

The full definition of the above algorithm identifiers and their corresponding parameters (an IV for block chaining) is provided in the CMS specification [8].

3.2. Public Key Authentication

Implementation of the changes in this section is REQUIRED for compliance with PKINIT.

3.2.1. Client Request

Public keys may be signed by some certification authority (CA), or they may be maintained by the KDC in which case the KDC is the trusted authority. Note that the latter mode does not require the use of certificates.

The initial authentication request is sent as per RFC 1510bis, except that a preauthentication field containing data signed by the user's private key accompanies the request:

```
PA-PK-AS-REQ ::= SEQUENCE {
                              -- PA TYPE 14
    signedAuthPack            [0] ContentInfo,
                                  -- Defined in CMS [8];
```

```
                                -- SignedData OID is {pkcs7 2}
                                -- AuthPack (below) defines the
                                -- data that is signed.
    trustedCertifiers       [1] SEQUENCE OF TrustedCas OPTIONAL,
                                -- This is a list of CAs that the
                                -- client trusts and that certify
                                -- KDCs.
    kdcCert                 [2] IssuerAndSerialNumber OPTIONAL
                                -- As defined in CMS [8];
                                -- specifies a particular KDC
                                -- certificate if the client
                                -- already has it.
    encryptionCert          [3] IssuerAndSerialNumber OPTIONAL
                                -- For example, this may be the
                                -- client's Diffie-Hellman
                                -- certificate, or it may be the
                                -- client's RSA encryption
                                -- certificate.
}

TrustedCas ::= CHOICE {
    principalName         [0] KerberosName,
                              -- as defined below
    caName                [1] Name
                              -- fully qualified X.500 name
                              -- as defined by X.509
    issuerAndSerial       [2] IssuerAndSerialNumber
                              -- Since a CA may have a number of
                              -- certificates, only one of which
                              -- a client trusts
}
```

The type of the ContentInfo in the signedAuthPack is SignedData. Its usage is as follows:

The SignedData data type is specified in the Cryptographic Message Syntax, a product of the S/MIME working group of the IETF. The following describes how to fill in the fields of this data:

1. The encapContentInfo field MUST contain the PKAuthenticator and, optionally, the client's Diffie Hellman public value.

   a. The eContentType field MUST contain the OID value for pkauthdata: iso (1) org (3) dod (6) internet (1) security (5) kerberosv5 (2) pkinit (3) pkauthdata (1)

   b. The eContent field is data of the type AuthPack (below).

2. The signerInfos field contains the signature of AuthPack.

3. The Certificates field, when non-empty, contains the client's certificate chain. If present, the KDC uses the public key from the client's certificate to verify the signature in the request. Note that the client may pass different certificate chains that are used for signing or for encrypting. Thus, the KDC may utilize a different client certificate for signature verification than the one it uses to encrypt the reply to the client. For example, the client may place a Diffie-Hellman certificate in this field in order to convey its static Diffie Hellman certificate to the KDC to enable static-ephemeral Diffie-Hellman mode for the reply; in this case, the client does NOT place its public value in the

AuthPack (defined below). As another example, the client may place an RSA encryption certificate in this field. However, there MUST always be (at least) a signature certificate.

4. When a DH key is being used, the public exponent is provided in the subjectPublicKey field of the SubjectPublicKeyInfo and the DH parameters are supplied as a DomainParameters in the AlgorithmIdentitfier parameters. The DH paramaters SHOULD be chosen from the First and Second defined Oakley Groups [The Internet Key Exchange (IKE) RFC-2409], if a server will not accept either of these groups, it will respond with a krb-error of KDC_ERR_KEY_TOO_WEAK and the e_data will contain a DomainParameters with appropriate parameters for the client to use.

5. The KDC may wish to use cached Diffie-Hellman parameters (see Section 3.2.2, KDC Response). To indicate acceptance of cached parameters, the client sends zero in the nonce field of the PKAuthenticator. Zero is not a valid value for this field under any other circumstances. If cached parameters are used, the client and the KDC MUST perform key derivation (for the appropriate cryptosystem) on the resulting encryption key, as specified in RFC 1510bis. (With a zero nonce, message binding is performed using the nonce in the main request, which must be encrypted using the encapsulated reply key.)

```
AuthPack ::= SEQUENCE {
    pkAuthenticator         [0] PKAuthenticator,
    clientPublicValue       [1] SubjectPublicKeyInfo OPTIONAL
                                -- if client is using Diffie-Hellman
                                -- (ephemeral-ephemeral only)
}

PKAuthenticator ::= SEQUENCE {
    cusec                   [0] INTEGER,
                                -- for replay prevention as in RFC 1510bis
    ctime                   [1] KerberosTime,
                                -- for replay prevention as in RFC 1510bis
    nonce                   [2] INTEGER,
                                -- zero only if client will accept
                                -- cached DH parameters from KDC;
                                -- must be non-zero otherwise
    pachecksum              [3] Checksum
                                -- Checksum over KDC-REQ-BODY
                                -- Defined by Kerberos spec;
                        -- must be unkeyed, e.g. sha1 or rsa-md5
}

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm                   AlgorithmIdentifier,
                                -- dhKeyAgreement
    subjectPublicKey            BIT STRING
                                -- for DH, equals
                                -- public exponent (INTEGER encoded
                                -- as payload of BIT STRING)
}   -- as specified by the X.509 recommendation [7]

AlgorithmIdentifier ::= SEQUENCE {
    algorithm                   OBJECT IDENTIFIER,
                                -- for dhKeyAgreement, this is
                                -- { iso (1) member-body (2) US (840)
                                -- ansi-x942(10046) number-type(2) 1 }
```

```
                                -- from RFC 2459 [12]
        parameters              ANY DEFINED by algorithm OPTIONAL
                                -- for dhKeyAgreement, this is
                                -- DomainParameters
    }   -- as specified by the X.509 recommendation [7]

    DomainParameters ::= SEQUENCE {
        p                       INTEGER, -- odd prime, p=jq +1
        g                       INTEGER, -- generator, g
        q                       INTEGER, -- factor of p-1
        j                       INTEGER OPTIONAL, -- subgroup factor
        validationParms         ValidationParms OPTIONAL
    }   -- as defined in RFC 2459 [12]

    ValidationParms ::= SEQUENCE {
        seed                    BIT STRING,
                                -- seed for the system parameter
                                -- generation process
        pgenCounter             INTEGER
                                -- integer value output as part
                                -- of the of the system parameter
                                -- prime generation process
    }   -- as defined in RFC 2459 [12]
```

If the client passes an issuer and serial number in the request, the KDC is requested to use the referred-to certificate. If none exists, then the KDC returns an error of type KDC_ERR_CERTIFICATE_MISMATCH. It also returns this error if, on the other hand, the client does not pass any trustedCertifiers, believing that it has the KDC's certificate, but the KDC has more than one certificate. The KDC should include information in the KRB-ERROR message that indicates the KDC certificate(s) that a client may utilize. This data is specified in the e-data, which is defined in RFC 1510bis revisions as a SEQUENCE of TypedData:

```
TypedData ::=  SEQUENCE {
                data-type      [0] INTEGER,
                data-value     [1] OCTET STRING,
} -- per Kerberos RFC 1510bis
```

where:
data-type = TD-PKINIT-CMS-CERTIFICATES = 101
data-value = CertificateSet // as specified by CMS [8]

The PKAuthenticator carries information to foil replay attacks, to bind the pre-authentication data to the KDC-REQ-BODY, and to bind the request and response. The PKAuthenticator is signed with the client's signature key.

3.2.2. KDC Response

Upon receipt of the AS_REQ with PA-PK-AS-REQ pre-authentication type, the KDC attempts to verify the user's certificate chain (userCert), if one is provided in the request. This is done by verifying the certification path against the KDC's policy of legitimate certifiers.

If the client's certificate chain contains no certificate signed by a CA trusted by the KDC, then the KDC sends back an error message of type KDC_ERR_CANT_VERIFY_CERTIFICATE. The accompanying e-data is a SEQUENCE of one TypedData (with type TD-TRUSTED-CERTIFIERS=104) whose data-value is an OCTET STRING which is the DER encoding of

```
    TrustedCertifiers ::= SEQUENCE OF PrincipalName
```

```
                        -- X.500 name encoded as a principal name
                        -- see Section 3.1
```

If while verifying a certificate chain the KDC determines that the signature on one of the certificates in the CertificateSet from the signedAuthPack fails verification, then the KDC returns an error of type KDC_ERR_INVALID_CERTIFICATE. The accompanying e-data is a SEQUENCE of one TypedData (with type TD-CERTIFICATE-INDEX=105) whose data-value is an OCTET STRING which is the DER encoding of the index into the CertificateSet ordered as sent by the client.

```
    CertificateIndex  ::= INTEGER
                        -- 0 = 1st certificate,
                        --     (in order of encoding)
                        -- 1 = 2nd certificate, etc
```

The KDC may also check whether any of the certificates in the client's chain has been revoked. If one of the certificates has been revoked, then the KDC returns an error of type KDC_ERR_REVOKED_CERTIFICATE; if such a query reveals that the certificate's revocation status is unknown or not available, then if required by policy, the KDC returns the appropriate error of type KDC_ERR_REVOCATION_STATUS_UNKNOWN or KDC_ERR_REVOCATION_STATUS_UNAVAILABLE. In any of these three cases, the affected certificate is identified by the accompanying e-data, which contains a CertificateIndex as described for KDC_ERR_INVALID_CERTIFICATE.

If the certificate chain can be verified, but the name of the client in the certificate does not match the client's name in the request, then the KDC returns an error of type KDC_ERR_CLIENT_NAME_MISMATCH. There is no accompanying e-data field in this case.

Even if all succeeds, the KDC may--for policy reasons--decide not to trust the client. In this case, the KDC returns an error message of type KDC_ERR_CLIENT_NOT_TRUSTED. One specific case of this is the presence or absence of an Enhanced Key Usage (EKU) OID within the certificate extensions. The rules regarding acceptability of an EKU sequence (or the absence of any sequence) are a matter of local policy. For the benefit of implementers, we define a PKINIT EKU OID as the following: iso (1) org (3) dod (6) internet (1) security (5) kerberosv5 (2) pkinit (3) pkekuoid (2).

If a trust relationship exists, the KDC then verifies the client's signature on AuthPack. If that fails, the KDC returns an error message of type KDC_ERR_INVALID_SIG. Otherwise, the KDC uses the timestamp (ctime and cusec) in the PKAuthenticator to assure that the request is not a replay. The KDC also verifies that its name is specified in the PKAuthenticator.

If the clientPublicValue field is filled in, indicating that the client wishes to use Diffie-Hellman key agreement, then the KDC checks to see that the parameters satisfy its policy. If they do not (e.g., the prime size is insufficient for the expected encryption type), then the KDC sends back an error message of type KDC_ERR_KEY_TOO_WEAK, with an e-data containing a structure of type DomainParameters with appropriate DH parameters for the client to retry the request. Otherwise, it generates its own public and private values for the response.

The KDC also checks that the timestamp in the PKAuthenticator is

within the allowable window and that the principal name and realm are correct. If the local (server) time and the client time in the authenticator differ by more than the allowable clock skew, then the KDC returns an error message of type KRB_AP_ERR_SKEW as defined in RFC 1510bis.

Assuming no errors, the KDC replies as per RFC 1510bis, except as follows. The user's name in the ticket is determined by the following decision algorithm:

1. If the KDC has a mapping from the name in the certificate to a Kerberos name, then use that name.
   Else
2. If the certificate contains the SubjectAltName extension and the local KDC policy defines a mapping from the SubjectAltName to a Kerberos name, then use that name.
   Else
3. Use the name as represented in the certificate, mapping as necessary (e.g., as per RFC 2253 for X.500 names). In this case the realm in the ticket MUST be the name of the certifier that issued the user's certificate.

Note that a principal name may be carried in the subjectAltName field of a certificate. This name may be mapped to a principal record in a security database based on local policy, for example the subjectAltName may be kerberos/principal@realm format. In this case the realm name is not that of the CA but that of the local realm doing the mapping (or some realm name chosen by that realm).

If a non-KDC X.509 certificate contains the principal name within the subjectAltName version 3 extension, that name may utilize KerberosName as defined below, or, in the case of an S/MIME certificate [14], may utilize the email address. If the KDC is presented with an S/MIME certificate, then the email address within subjectAltName will be interpreted as a principal and realm separated by the "@" sign, or as a name that needs to be mapped according to local policy. If the resulting name does not correspond to a registered principal name, then the principal name is formed as defined in section 3.1.

The trustedCertifiers field contains a list of certification authorities trusted by the client, in the case that the client does not possess the KDC's public key certificate. If the KDC has no certificate signed by any of the trustedCertifiers, then it returns an error of type KDC_ERR_KDC_NOT_TRUSTED.

KDCs should try to (in order of preference):
1. Use the KDC certificate identified by the serialNumber included in the client's request.
2. Use a certificate issued to the KDC by one of the client's trustedCertifier(s);

If the KDC is unable to comply with any of these options, then the KDC returns an error message of type KDC_ERR_KDC_NOT_TRUSTED to the client.

The KDC encrypts the reply not with the user's long-term key, but with the Diffie Hellman derived key or a random key generated for this particular response which is carried in the padata field of the TGS-REP message.

```
PA-PK-AS-REP ::= CHOICE {
                        -- PA TYPE 15
```

```
    dhSignedData       [0] ContentInfo,
                        -- Defined in CMS [8] and used only with
                        -- Diffie-Hellman key exchange (if the
                        -- client public value was present in the
                        -- request).
                        -- SignedData OID is {pkcs7 2}
                        -- This choice MUST be supported
                        -- by compliant implementations.
    encKeyPack         [1] ContentInfo
                        -- Defined in CMS [8].
                        -- The temporary key is encrypted
                        -- using the client public key
                        -- key.
                        -- EnvelopedData OID is {pkcs7 3}
                        -- SignedReplyKeyPack, encrypted
                        -- with the temporary key, is also
                        -- included.
}
```

The type of the ContentInfo in the dhSignedData is SignedData. Its usage is as follows:

When the Diffie-Hellman option is used, dhSignedData in PA-PK-AS-REP provides authenticated Diffie-Hellman parameters of the KDC. The reply key used to encrypt part of the KDC reply message is derived from the Diffie-Hellman exchange:

1. Both the KDC and the client calculate a secret value (g^ab mod p), where a is the client's private exponent and b is the KDC's private exponent.

2. Both the KDC and the client take the first N bits of this secret value and convert it into a reply key. N depends on the reply key type.

   a. For example, if the reply key is DES, N=64 bits, where some of the bits are replaced with parity bits, according to FIPS PUB 74.

   b. As another example, if the reply key is (3-key) 3-DES, N=192 bits, where some of the bits are replaced with parity bits, according to FIPS PUB 74.

3. The encapContentInfo field MUST contain the KdcDHKeyInfo as defined below.

   a. The eContentType field MUST contain the OID value for pkdhkeydata: iso (1) org (3) dod (6) internet (1) security (5) kerberosv5 (2) pkinit (3) pkdhkeydata (2)

   b. The eContent field is data of the type KdcDHKeyInfo (below).

4. The certificates field MUST contain the certificates necessary for the client to establish trust in the KDC's certificate based on the list of trusted certifiers sent by the client in the PA-PK-AS-REQ. This field may be empty if the client did not send to the KDC a list of trusted certifiers (the trustedCertifiers field was empty, meaning that the client already possesses the KDC's certificate).

5. The signerInfos field is a SET that MUST contain at least one member, since it contains the actual signature.

6. If the client indicated acceptance of cached Diffie-Hellman parameters from the KDC, and the KDC supports such an option (for performance reasons), the KDC should return a zero in the nonce field and include the expiration time of the parameters in the dhKeyExpiration field. If this time is exceeded, the client SHOULD NOT use the reply. If the time is absent, the client SHOULD NOT use the reply and MAY resubmit a request with a non-zero nonce (thus indicating non-acceptance of cached Diffie-Hellman parameters). As indicated above in Section 3.2.1, Client Request, when the KDC uses cached parameters, the client and the KDC MUST perform key derivation (for the appropriate cryptosystem) on the resulting encryption key, as specified in RFC 1510bis.

```
KdcDHKeyInfo ::= SEQUENCE {
                          -- used only when utilizing Diffie-Hellman
  subjectPublicKey      [0] BIT STRING,
                            -- Equals public exponent (g^a mod p)
                            -- INTEGER encoded as payload of
                            -- BIT STRING
  nonce                 [1] INTEGER,
                            -- Binds response to the request
                            -- Exception: Set to zero when KDC
                            -- is using a cached DH value
  dhKeyExpiration       [2] KerberosTime OPTIONAL
                            -- Expiration time for KDC's cached
                            -- DH value
}
```

The type of the ContentInfo in the encKeyPack is EnvelopedData. Its usage is as follows:

The EnvelopedData data type is specified in the Cryptographic Message Syntax, a product of the S/MIME working group of the IETF. It contains a temporary key encrypted with the PKINIT client's public key. It also contains a signed and encrypted reply key.

1. The originatorInfo field is not required, since that information may be presented in the signedData structure that is encrypted within the encryptedContentInfo field.

2. The optional unprotectedAttrs field is not required for PKINIT.

3. The recipientInfos field is a SET which MUST contain exactly one member of the KeyTransRecipientInfo type for encryption with a public key.

   a. The encryptedKey field (in KeyTransRecipientInfo) contains the temporary key which is encrypted with the PKINIT client's public key.

4. The encryptedContentInfo field contains the signed and encrypted reply key.

   a. The contentType field MUST contain the OID value for id-signedData: iso (1) member-body (2) us (840) rsadsi (113549) pkcs (1) pkcs7 (7) signedData (2)

   b. The encryptedContent field is encrypted data of the CMS type signedData as specified below.

i. The encapContentInfo field MUST contains the ReplyKeyPack.

* The eContentType field MUST contain the OID value for pkrkeydata: iso (1) org (3) dod (6) internet (1) security (5) kerberosv5 (2) pkinit (3) pkrkeydata (3)

* The eContent field is data of the type ReplyKeyPack (below).

ii. The certificates field MUST contain the certificates necessary for the client to establish trust in the KDC's certificate based on the list of trusted certifiers sent by the client in the PA-PK-AS-REQ. This field may be empty if the client did not send to the KDC a list of trusted certifiers (the trustedCertifiers field was empty, meaning that the client already possesses the KDC's certificate).

iii. The signerInfos field is a SET that MUST contain at least one member, since it contains the actual signature.

```
ReplyKeyPack ::= SEQUENCE {
                        -- not used for Diffie-Hellman
    replyKey            [0] EncryptionKey,
                            -- from RFC 1510bis
                            -- used to encrypt main reply
                            -- ENCTYPE is at least as strong as
                            -- ENCTYPE of session key
    nonce               [1] INTEGER,
                            -- binds response to the request
                            -- must be same as the nonce
                            -- passed in the PKAuthenticator
}
```

3.2.2.1. Use of transited Field

Since each certifier in the certification path of a user's certificate is equivalent to a separate Kerberos realm, the name of each certifier in the certificate chain MUST be added to the transited field of the ticket. The format of these realm names is defined in Section 3.1 of this document. If applicable, the transit-policy-checked flag should be set in the issued ticket.

3.2.2.2. Kerberos Names in Certificates

The KDC's certificate(s) MUST bind the public key(s) of the KDC to a name derivable from the name of the realm for that KDC. X.509 certificates MUST contain the principal name of the KDC (defined in RFC 1510bis) as the SubjectAltName version 3 extension. Below is the definition of this version 3 extension, as specified by the X.509 standard:

```
    subjectAltName EXTENSION ::= {
        SYNTAX GeneralNames
        IDENTIFIED BY id-ce-subjectAltName
    }

    GeneralNames ::= SEQUENCE SIZE(1..MAX) OF GeneralName
```

```
    GeneralName ::= CHOICE {
        otherName       [0] OtherName,
        ...
    }

    OtherName ::= SEQUENCE {
        type-id         OBJECT IDENTIFIER,
        value           [0] EXPLICIT ANY DEFINED BY type-id
    }
```

For the purpose of specifying a Kerberos principal name, the value in OtherName MUST be a KerberosName, defined as follows:

```
    KerberosName ::= SEQUENCE {
        realm           [0] Realm,
        principalName   [1] PrincipalName
    }
```

This specific syntax is identified within subjectAltName by setting the type-id in OtherName to krb5PrincipalName, where (from the Kerberos specification) we have

```
    krb5 OBJECT IDENTIFIER ::= { iso (1)
                                 org (3)
                                 dod (6)
                                 internet (1)
                                 security (5)
                                 kerberosv5 (2) }

    krb5PrincipalName OBJECT IDENTIFIER ::= { krb5 2 }
```

(This specification may also be used to specify a Kerberos name within the user's certificate.) The KDC's certificate may be signed directly by a CA, or there may be intermediaries if the server resides within a large organization, or it may be unsigned if the client indicates possession (and trust) of the KDC's certificate.

Note that the KDC's principal name has the instance equal to the realm, and those fields should be appropriately set in the realm and principalName fields of the KerberosName. This is the case even when obtaining a cross-realm ticket using PKINIT.

3.2.3. Client Extraction of Reply

The client then extracts the random key used to encrypt the main reply. This random key (in encPaReply) is encrypted with either the client's public key or with a key derived from the DH values exchanged between the client and the KDC. The client uses this random key to decrypt the main reply, and subsequently proceeds as described in RFC 1510bis.

3.2.4. Required Algorithms

Not all of the algorithms in the PKINIT protocol specification have to be implemented in order to comply with the proposed standard. Below is a list of the required algorithms:

* Diffie-Hellman public/private key pairs
    * utilizing Diffie-Hellman ephemeral-ephemeral mode
* SHA1 digest and RSA for signatures
* SHA1 digest for the Checksum in the PKAuthenticator

```
    * using Kerberos checksum type 'sha1'
* 3-key triple DES keys derived from the Diffie-Hellman Exchange
* 3-key triple DES Temporary and Reply keys
```

4. Logistics and Policy

This section describes a way to define the policy on the use of PKINIT for each principal and request.

The KDC is not required to contain a database record for users who use public key authentication. However, if these users are registered with the KDC, it is recommended that the database record for these users be modified to an additional flag in the attributes field to indicate that the user should authenticate using PKINIT. If this flag is set and a request message does not contain the PKINIT preauthentication field, then the KDC sends back as error of type KDC_ERR_PREAUTH_REQUIRED indicating that a preauthentication field of type PA-PK-AS-REQ must be included in the request.

5. Security Considerations

PKINIT raises a few security considerations, which we will address in this section.

First of all, PKINIT extends the cross-realm model to the public key infrastructure. Anyone using PKINIT must be aware of how the certification infrastructure they are linking to works.

Also, as in standard Kerberos, PKINIT presents the possibility of interactions between different cryptosystems of varying strengths, and this now includes public-key cryptosystems. Many systems, for instance, allow the use of 512-bit public keys. Using such keys to wrap data encrypted under strong conventional cryptosystems, such as triple-DES, may be inappropriate.

Care should be taken in how certificates are chosen for the purposes of authentication using PKINIT. Some local policies require that key escrow be applied for certain certificate types. People deploying PKINIT should be aware of the implications of using certificates that have escrowed keys for the purposes of authentication.

As described in Section 3.2, PKINIT allows for the caching of the Diffie-Hellman parameters on the KDC side, for performance reasons. For similar reasons, the signed data in this case does not vary from message to message, until the cached parameters expire. Because of the persistence of these parameters, the client and the KDC are to use the appropriate key derivation measures (as described in RFC 1510bis) when using cached DH parameters.

PKINIT does not provide for a "return routability test" to prevent attackers from mounting a denial of service attack on the KDC by causing it to perform needless expensive cryptographic operations. Strictly speaking, this is also true of base Kerberos, although the potential cost is not as great in base Kerberos, because it does not make use of public key cryptography.

Lastly, PKINIT calls for randomly generated keys for conventional cryptosystems. Many such systems contain systematically "weak" keys. For recommendations regarding these weak keys, see RFC 1510bis.

6. Transport Issues

Certificate chains can potentially grow quite large and span several UDP packets; this in turn increases the probability that a Kerberos message involving PKINIT extensions will be broken in transit. In light of the possibility that the Kerberos specification will require KDCs to accept requests using TCP as a transport mechanism, we make the same recommendation with respect to the PKINIT extensions as well.

7. Bibliography

[1] J. Kohl, C. Neuman. The Kerberos Network Authentication Service (V5). Request for Comments 1510.

[2] B.C. Neuman, Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994.

[3] M. Sirbu, J. Chuang. Distributed Authentication in Kerberos Using Public Key Cryptography. Symposium On Network and Distributed System Security, 1997.

[4] B. Cox, J.D. Tygar, M. Sirbu. NetBill Security and Transaction Protocol. In Proceedings of the USENIX Workshop on Electronic Commerce, July 1995.

[5] T. Dierks, C. Allen. The TLS Protocol, Version 1.0 Request for Comments 2246, January 1999.

[6] B.C. Neuman, Proxy-Based Authorization and Accounting for Distributed Systems. In Proceedings of the 13th International Conference on Distributed Computing Systems, May 1993.

[7] ITU-T (formerly CCITT) Information technology - Open Systems Interconnection - The Directory: Authentication Framework Recommendation X.509 ISO/IEC 9594-8

[8] R. Housley. Cryptographic Message Syntax. draft-ietf-smime-cms-13.txt, April 1999, approved for publication as RFC.

[9] PKCS #7: Cryptographic Message Syntax Standard, An RSA Laboratories Technical Note Version 1.5 Revised November 1, 1993

[10] R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc. A Description of the RC2(r) Encryption Algorithm March 1998. Request for Comments 2268.

[11] M. Wahl, S. Kille, T. Howes. Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names. Request for Comments 2253.

[12] R. Housley, W. Ford, W. Polk, D. Solo. Internet X.509 Public Key Infrastructure, Certificate and CRL Profile, January 1999. Request for Comments 2459.

[13] B. Kaliski, J. Staddon. PKCS #1: RSA Cryptography Specifications, October 1998. Request for Comments 2437.

[14] S. Dusse, P. Hoffman, B. Ramsdell, J. Weinstein. S/MIME Version 2 Certificate Handling, March 1998. Request for Comments 2312.

[15] M. Wahl, T. Howes, S. Kille. Lightweight Directory Access Protocol (v3), December 1997. Request for Comments 2251.

[16] ITU-T (formerly CCITT) Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1) Rec. X.680 ISO/IEC 8824-1

[17] PKCS #3: Diffie-Hellman Key-Agreement Standard, An RSA Laboratories Technical Note, Version 1.4, Revised November 1, 1993.

8. Acknowledgements

Some of the ideas on which this proposal is based arose during discussions over several years between members of the SAAG, the IETF CAT working group, and the PSRG, regarding integration of Kerberos and SPX. Some ideas have also been drawn from the DASS system. These changes are by no means endorsed by these groups. This is an attempt to revive some of the goals of those groups, and this proposal approaches those goals primarily from the Kerberos perspective. Lastly, comments from groups working on similar ideas in DCE have been invaluable.

9. Expiration Date

This draft expires June 25, 2002.

10. Authors

Brian Tung
Clifford Neuman
USC Information Sciences Institute
4676 Admiralty Way Suite 1001
Marina del Rey CA 90292-6695
Phone: +1 310 822 1511
E-mail: {brian, bcn}@isi.edu

Matthew Hur
Cisco Systems
2901 Third Avenue
Seattle WA 98121
Phone: (206) 256-3197
E-Mail: mhur@cisco.com

Ari Medvinsky
Keen.com, Inc.
150 Independence Drive
Menlo Park CA 94025
Phone: +1 650 289 3134
E-mail: ari@keen.com

Sasha Medvinsky
Motorola
6450 Sequence Drive
San Diego, CA 92121
+1 858 404 2367
E-mail: smedvinsky@gi.com

John Wray
Iris Associates, Inc.
5 Technology Park Dr.
Westford, MA 01886
E-mail: John_Wray@iris.com

Jonathan Trostle
Cisco Systems
170 W. Tasman Dr.
San Jose, CA 95134
E-mail: jtrostle@cisco.com

Appendix D. Example of MMH Algorithm Implementation (Informative)

This appendix gives an example implementation of the MMH MAC algorithm. There may be other implementations that have advantages over this example in particular operating environments. This example is for informational purposes only and is meant to clarify the specification.

The example implementation uses the term “MMH16” for the case where the MAC length is 2 octets and “MMH32” for the case where the length is 4 octets.

A main program is included for exercising the example implementation. The output produced by the program is included.

```
/*
 Demo of PacketCable MMH16 and MMH32 MAC algorithms.

 This program has been tested using Microsoft C/C++ Version 5.0.
 It is believed to port easily to other compilers, but this has
 not been tested. When porting, be sure to pick the definitions
 for int16, int32, uint16, and uint32 carefully.
*/

#include <stdio.h>

/*
Define signed and unsigned integers having 16 and 32 bits.
This is machine/compiler dependent, so pick carefully.
*/
typedef short int16;
typedef unsigned short uint16;
typedef int int32;
typedef unsigned int uint32;

/*
Define this symbol to see intermediate values.
Comment it out for clean display.
*/
#define VERBOSE

int32 reduceModF4(int32 x) {


 /*
 Routine to reduce an int32 value modulo F4, where F4 = 0x10001.
 Result is in range [0, 0x10000].
 */


 int32 xHi, xLo;

 /* Range of x is [0x80000000, 0x7fffffff]. */

 /*
 If x is negative, add a multiple of F4 to make it non-negative.
 This loop executes no more than two times.
 */
 while (x < 0) x += 0x7fff7fff;

 /* Range of x is [0, 0x7fffffff]. */

 /* Subtract high 16 bits of x from low 16 bits. */
```

```
 xHi = x >> 16;
 xLo = x & 0xffff;
 x = xLo - xHi;

 /* Range of x is [0xffff8001, 0x0000ffff]. */

 /* If x is negative, add F4. */
 if (x < 0) x += 0x10001;

 /* Range of x is [0, 0x10000]. */

 return x;
}

uint16 mmh16(
 unsigned char *message,
 unsigned char *key,
 unsigned char *pad,
 int msgLen
) {

 /*
 Compute and return the MMH16 MAC of the message using the
 indicated key and pad.

 The length of the message is msgLen bytes; msgLen must be even.

 The length of the key must be at least msgLen bytes.

 The length of the pad is two bytes. The pad must be freshly
 picked from a secure random source.
 */

 int16 x, y;
 uint16 u, v;
 int32 sum;
 int i;

 sum = 0;

 for (i=0; i<msgLen; i+=2) {

 /* Build a 16-bit factor from the next two message bytes. */
 x = *message++;
 x <<= 8;
 x |= *message++;

 /* Build a 16-bit factor from the next two key bytes. */
 y = *key++;
 y <<= 8;
 y |= *key++;

 /* Accumulate product of the factors into 32-bit sum */
 sum += (int32)x * (int32)y;

 #ifdef VERBOSE
 printf(" x %04x y %04x sum %08x\n", x & 0xffff, y & 0xffff, sum);
 #endif

 }

 /* Reduce sum modulo F4 and truncate to 16 bits. */
 u = (uint16) reduceModF4(sum);
```

```
 #ifdef VERBOSE
 printf(" sum mod F4, truncated to 16 bits: %04x\n", u & 0xffff);
 #endif

 /* Build the pad variable from the two pad bytes */
 v = *pad++;
 v <<= 8;
 v |= *pad;

 #ifdef VERBOSE
 printf(" pad variable: %04x\n", v & 0xffff);
 #endif

 /* Accumulate pad variable, truncate to 16 bits */
 u = (uint16)(u + v);

 #ifdef VERBOSE
 printf(" mmh16 value: %04x\n", u & 0xffff);
 #endif

 return u;
}

uint32 mmh32(
 unsigned char *message,
 unsigned char *key,
 unsigned char *pad,
 int msgLen
) {

 /*
 Compute and return the MMH32 MAC of the message using the
 indicated key and pad.

 The length of the message is msgLen bytes; msgLen must be even.

 The length of the key must be at least (msgLen + 2) bytes.

 The length of the pad is four bytes. The pad must be freshly
 picked from a secure random source.
 */

 uint16 x, y;
 uint32 sum;

 x = mmh16(message, key, pad, msgLen);
 y = mmh16(message, key+2, pad+2, msgLen);
 sum = x;
 sum <<= 16;
 sum |= y;

 return sum;
}


void show(char *name, unsigned char *src, int nbytes) {


 /*
 Routine to display a byte array, in normal or reverse order
 */
```

```
 int i;
 enum {
 BYTES_PER_LINE = 16
 };

 if (name) printf("%s", name);

 for (i=0; i<nbytes; i++) {
 if ((i % BYTES_PER_LINE) == 0) printf("\n");
 printf("%02x ", src[i]);
 }
 printf("\n");
}

int main() {

 uint16 mac16;
 uint32 mac32;

 unsigned char message[] = {
 0x4e, 0x6f, 0x77, 0x20, 0x69, 0x73, 0x20, 0x74, 0x68,
 0x65, 0x20, 0x74, 0x69, 0x6d, 0x65, 0x2e,
 };

 unsigned char key[] = {
 0x35, 0x2c, 0xcf, 0x84, 0x95, 0xef, 0xd7, 0xdf, 0xb8,
 0xf5, 0x74, 0x05, 0x95, 0xeb, 0x98, 0xd6, 0xeb, 0x98,
 };

 unsigned char pad16[] = {
 0xae, 0x07,
 };

 unsigned char pad32[] = {
 0xbd, 0xe1, 0x89, 0x7b,
 };

 unsigned char macBuf[4];


 printf("Example of MMH16 computation\n");
 show("message", message, sizeof(message));
 show("key", key, sizeof(message));
 show("pad", pad16, 2);

 mac16 = mmh16(message, key, pad16, sizeof(message));
 macBuf[1] = (unsigned char)mac16; mac16 >>= 8;
 macBuf[0] = (unsigned char)mac16;

 show("MMH16 MAC", macBuf, 2);
 printf("\n");

 printf("Example of MMH32 computation\n");
 show("message", message, sizeof(message));
 show("key", key, sizeof(message)+2);
 show("pad", pad32, 4);

 mac32 = mmh32(message, key, pad32, sizeof(message));
 macBuf[3] = (unsigned char)mac32; mac32 >>= 8;
 macBuf[2] = (unsigned char)mac32; mac32 >>= 8;
 macBuf[1] = (unsigned char)mac32; mac32 >>= 8;
 macBuf[0] = (unsigned char)mac32;
```

```
 show("MMH32 MAC", macBuf, 4);
 printf("\n");

 return 0;
}
```

Here is the output produced by the program:

```
Example of MMH16 computation
message
4e 6f 77 20 69 73 20 74 68 65 20 74 69 6d 65 2e
key
35 2c cf 84 95 ef d7 df b8 f5 74 05 95 eb 98 d6
pad
ae 07
 x 4e6f y 352c sum 104a7614
 x 7720 y cf84 sum f9bac294
 x 6973 y 95ef sum ce0a23f1
 x 2074 y d7df sum c8f3d4fd
 x 6865 y b8f5 sum abfb55a6
 x 2074 y 7405 sum bab087ea
 x 696d y 95eb sum 8f00bff9
 x 652e y 98d6 sum 663aa46d
 sum mod F4, truncated to 16 bits: 3e33
 pad variable: ae07
 mmh16 value: ec3a
MMH16 MAC
ec 3a

Example of MMH32 computation
message
4e 6f 77 20 69 73 20 74 68 65 20 74 69 6d 65 2e
key
35 2c cf 84 95 ef d7 df b8 f5 74 05 95 eb 98 d6
eb 98
pad
bd e1 89 7b
 x 4e6f y 352c sum 104a7614
 x 7720 y cf84 sum f9bac294
 x 6973 y 95ef sum ce0a23f1
 x 2074 y d7df sum c8f3d4fd
 x 6865 y b8f5 sum abfb55a6
 x 2074 y 7405 sum bab087ea
 x 696d y 95eb sum 8f00bff9
 x 652e y 98d6 sum 663aa46d
 sum mod F4, truncated to 16 bits: 3e33
 pad variable: bde1
 mmh16 value: fc14
 x 4e6f y cf84 sum f125323c
 x 7720 y 95ef sum bfca091c
 x 6973 y d7df sum af427949
 x 2074 y b8f5 sum a640e84d
 x 6865 y 7405 sum d590b646
 x 2074 y 95eb sum c81e04c2
 x 696d y 98d6 sum 9da1dde0
 x 652e y eb98 sum 95912b30
 sum mod F4, truncated to 16 bits: 959f
 pad variable: 897b
 mmh16 value: 1f1a
MMH32 MAC
fc 14 1f 1a
```

Appendix E. Oakley Groups

PKINIT states that DH parameters SHOULD be taken from the first or second Oakley groups as defined in [26]. Additionally, this specification requires that DH groups are used exactly as defined in [26].

[26] defines several so-called “Oakley groups.” Only the first two are relevant to this specification. [26] requires implementations to support the first group, and recommends that they support the second. This Appendix is included because [26] does not give values of q (the p-1 factor) for the groups, and these are necessary in order to encode the dhpublicnumber type used in the subjectPublicKeyInfo data structure in PKINIT.

The first two Oakley groups are defined as follows:

First Oakley Group:

Prime (p):

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A63A3620 FFFFFFFF FFFFFFFF
```

Generator (g or b):

2.

Factor (q):

```
7FFFFFFF FFFFFFFF E487ED51 10B4611A 62633145 C06E0E68
94812704 4533E63A 0105DF53 1D89CD91 28A5043C C71A026E
F7CA8CD9 E69D218D 98158536 F92F8A1B A7F09AB6 B6A8E122
F242DABB 312F3F63 7A262174 D31D1B10 7FFFFFFF FFFFFFFF
```

Second Oakley Group:

Prime (p):

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE65381
FFFFFFFF FFFFFFFF
```

Generator (g or b):

2.

Factor (q):

```
7FFFFFFF FFFFFFFF E487ED51 10B4611A 62633145 C06E0E68
94812704 4533E63A 0105DF53 1D89CD91 28A5043C C71A026E
F7CA8CD9 E69D218D 98158536 F92F8A1B A7F09AB6 B6A8E122
F242DABB 312F3F63 7A262174 D31BF6B5 85FFAE5B 7A035BF6
F71C35FD AD44CFD2 D74F9208 BE258FF3 24943328 F67329C0
FFFFFFFF FFFFFFFF
```

Appendix F. Acknowledgements

CableLabs and its participating member companies would like to extend a heartfelt thanks to all those who contributed to the development of this specification. All the participants of the Security focus team have added value to this effort through their participation.

Sasha Medvinsky, Rick Vetter (Motorola)

Matthew Broda and Louis LeVay (Nortel Networks)

Doc Evans (Cadant)

Derek Atkins (Telcordia)

Michael Thomas, Peter Grossman, Jonathan Trostle, Jan Vilhuber and Flemming Andreasen (Cisco)

Itay Sherman, Roy Spitzer, Satish Kumar and Manjuprakish Rao (Texas Instruments)

Brian Hagar (AG Communications Systems)

Ali Negahdar, Bill Tang, Rick Morris and Jiri Matousek (Arris Interactive)

Chris Melle, Bill Aiello, Bill Marshall and Steve Bellovin (AT&T)

Jay Xia (Nuera)

Sumanth Channabasappa (Alopa)

Matt Hur and Mike Froh (CyberSafe)

Jeff Carr, Eugene Nechamkin, Jim Shoghli (Broadcom)

Mike St. Johns (Excite@Home)

Alex Deacon (VeriSign)

Basant Dhakal (Tellabs)

Matt Osman, Nancy Davoust (CableLabs)

Louis LeVay (Nortel Networks)

Eric Rosenfeld (CableLabs)

Stuart Hoggan (CableLabs Team Lead)

Appendix G. Revision History

The following Engineering Changes have been incorporated in PKT-SP-SEC1.5-I02-070412.

ECS Name	Date Approved	Summary
SEC1.5-N-06-0314-1	3/13/2006	Fix TLS Certificate Reference
SEC1.5-N-07-0399.2	3/12/2007	Clarification of eCM DOCSIS versions
SEC1.5-N-07.0392-3	3/12/2007	Incorporation of support for IETF MIB modules

The following Engineering Change has been incorporated in PKT-SP-SEC1.5-I03-090624.

ECS Name	Date Approved	Summary
SEC1.5-N-09.0566-1	6/8/09	IKE SA Renewal Clarification