PacketCable™ Security Specification

PKT-SP-SEC-I02-001229

Document Status Sheet

Document Control Number:	PKT-SP-SEC-I02-001229			
Document Title:	PacketCable™ Security Specification			
Revision History:	I01 — First Interim release December 01, 1999			
	I02 — Second Interim release December 29, 2000			
Date:	December 29, 2000			
Status:	~~Work in Progress~~	~~Draft~~	Interim	~~Released~~
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.
Interim	A document, which has undergone rigorous Member and vendor review, suitable for use by vendors to design in conformance to and for field-testing. For purposes of the "Contribution and License Agreement for Intellectual Property" which grants licenses to the intellectual property contained in the PacketCable Specification, an "Interim Specification" is a "Published" Specification.
Released	A stable document, reviewed, tested and validated, suitable to enable cross-vendor interoperability.

Contents

Figures

Tables

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 will 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 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 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 as 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 Technical Report [1] and PacketCable 1.2Architecture and Technical Report [2] defines the overall PacketCable architecture, 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 architecture, 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 MTAs are within the current scope. Standalone MTAs will be addressed in later phases and security issues for Standalone MTAs are outside the scope of this document.
- 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; therefore 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 handing. Record Keeping Servers (RKS), Network Management Systems, File Transfer Protocol (FTP) servers and Dynamic Host Control Protocol (DHCP) servers are all considered to be unique to any service providers' implementation and are beyond the scope of this specification.

1.2.3 Requirements

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

- All MTAs must use DOCSIS 1.1 compliant cable modems and must implement BPI+.

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 weighted 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.

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 1.0 Architecture and PacketCable 1.2 Technical Reports [1][2] 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 Upgrade. This section specifies the secure loading and upgrading of software to the MTAs.

2 REFERENCES

Normative references companion standards and specifications; *informative* references background material.

2.1 Normative References

[1] *PacketCable 1.0 Architecture Framework Technical Report*, PKT-TR-ARCH-I01-991201, December 1, 1999, Cable Television Laboratories, Inc., http://www.PacketCable.com./

[2] *PacketCable 1.2 Architecture Technical Report*, PKT-TR-ARCH1.2-V01-001231, November 22, 2000, Cable Television Laboratories, Inc. http://www.PacketCable.com./

[3] *PacketCable Network-Based Call Signaling Protocol Specification*, PKT-SP-EC-MGCP-I02-991201, December 1, 1999, Cable Television Laboratories, Inc., http://www.PacketCable.com./

[4] *PacketCable Dynamic Quality of Service Specification*, PKT-SP-DQOS-I01-991201, December 1, 1999, Cable Television Laboratories, Inc., http://www.PacketCable.com./

[5] *PacketCable Interdomain Quality of Service Specification*, PKT-SP-IQOS-D01-000918, September 18, 2000, Cable Television Laboratories, Inc., http://www.PacketCable.com./

[6] *PacketCable PSTN Gateway Architecture Description and Functional Requirements*, PKT-FR-PSTNGWARCH-D01-990305, March 5, 1999, Cable Television Laboratories, Inc. http://www.PacketCable.com./

[7] *PacketCable MTA Device Provisioning Specification*, PKT-SP-PROV-I01-991201, December 1, 1999, Cable Television Laboratories, Inc., http://www.PacketCable.com./

[8] *PacketCable Event Messages*, PKT-SP-EM-I01-991201, December 1, 1999, Cable Television Laboratories, Inc., http://www.PacketCable.com./

[9] *PacketCable OSS Overview*, PKT-TR-OSS-I01-991201, December 1, 1999, Cable Television Laboratories, Inc., http://www.PacketCable.com./

[10] *PacketCable Audio/Video Codecs Specification*, PKT-SP-CODEC-I01-991201, December 1, 1999, Cable Television Laboratories, Inc., http://www.PacketCable.com./

[11] *PacketCable Electronic Surveillance Specification, pkt-sp-esp-i01-991229, December 29, 1999.*

[12] *Data-Over-Cable Service Interface Specifications, Radio Frequency Interface Specification*, SP-RFIv1.1-I03-991105, Cable Television Laboratories, Inc., November 05, 1999. http://www.CableLabs.com/

[13] *IEEE 802.14a First Working Group Ballot 30 High Capacity Physical Layer Specification*, Draft 1, Revision 3, March 1999.

[14] *Data-Over-Cable Service Interface Specifications, Baseline Privacy Plus Interface Specification*, SP-BPI+-I01-990316, Cable Television Laboratories, Inc., March 16, 1999. http://www.CableLabs.com/

[15] *RTP: A Transport Protocol for Real-Time Applications*, IETF (H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson), Internet Proposed Standard, RFC 1889, January, 1996.

[16] *RSVP Cryptographic Authentication*, IETF (F. Baker, B. Lindell, M. Talwar), Internet Draft draft-ietf-rsvp-md5-08.txt, expires in August, 1999.

[17] *HMAC: Keyed-Hashing for Message Authentication*, IETF (Krawczyk, Bellare, and Canetti), Internet Proposed Standard, RFC 2104, March 1996.

[18] *The Kerberos Network Authentication Service (V5)*, IETF Draft, Clifford Neuman, John Kohl, Theodore Ts'o, http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-revisions-04.txt, July, 1999.

[19] *Public Key Cryptography for Initial Authentication in Kerberos*, IETF Draft (B. Tung, C. Neuman, J. Wray, A. Medvinsky, M. Hur, S. Medvinsky, J. Trostle), http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-init-09.txt, July, 1999.

[20] *Cryptographic Message Syntax*, IETF (R. Housley), Internet Proposed Standard, RFC 2630, June 1999.

[21] *RADIUS Accounting*, IETF (C. Rigney), Internet Proposed Standard, RFC 2139, April 1997.

[22] *SDP: Session Description Protocol*, IETF (M. Handley, V. Jacobson), Internet Proposed Standard, RFC 2327, April 1998.

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

[24] *PKCS #1: RSA Encryption Standard.* Version 1.5, RSA Laboratories, November, 1993.

[25] *PKCS #1: RSA Encryption Standard.* Version 2, RSA Laboratories, September, 1998.

[26] *PKCS #7: Cryptographic Message Syntax Standard*, RSA Laboratories, November, 1993.

[27] *The TLS Protocol Version 1.0*, IETF (T. Dierks, C. Allen), Internet Proposed Standard, RFC 2246, January 1999.

[28] 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-Verloag, pp. 172-189

[29] *Security Architecture for the Internet Protocol*, IETF (S. Kent, R. Atkinson), Internet Proposed Standard, RFC 2401, November 1998.

[30] *IP Encapsulating Security Payload (ESP)*, IETF (D. Piper), Internet Proposed Standard, RFC 2406, November 1998.

[31] *The Internet IP Security Domain of Interpretation for ISAKMP*, IETF (S. Kent, R. Atkinson), Internet Proposed Standard, RFC 2407, November 1998.

[32] *The ESP CBC-Mode Cipher Algorithms*, IETF (R. Pereira, R. Adams), Internet Proposed Standard, RFC 2451, November 1998.

[33] *The Use of HMAC-SHA-1-96 within ESP and AH*, IETF (C. Madson, R. Glenn), Internet Proposed Standard, RFC 2404, November 1998.

[34] *The Internet Key Exchange (IKE)*, IETF (D. Harkins, D. Carrel), Internet Proposed Standard, RFC 2409, November 1998.

[35] *PacketCable MTA MIB*, PKT-SP-MIBS-MTA-I01-991201, Cable Television Laboratories, Inc., December 1, 1999. http://www.PacketCable.com./

[36] *PacketCable NCS MIB*, PKT-SP-MIBS-NCS-I01-991201, Cable Television Laboratories, Inc., December 1, 1999. http://www.PacketCable.com./

[37] *PacketCable MIB Framework*, PKT-SP-MIBS-I01-991201, Cable Television Laboratories, Inc., December 1, 1999. http://www.PacketCable.com./

[38] *PacketCable Audio Server Protocol*, PKT-SP-ASP-D01-000918, September 18, 2000, Cable Television Laboratories, Inc. http://www.PacketCable.com./

[39] *PacketCable PSTN Gateway Call Signaling Protocol Specification*, PKT-SP-TGCP-I01-991201, December 1, 1999, Cable Television Laboratories, Inc., http://www.PacketCable.com./

[40] *PacketCable Internet Signaling Transport Protocol (ISTP)*, PKT-SP-ISTP-I01-991201, December 1, 1999, Cable Television Laboratories, Inc., http://www.PacketCable.com./,

[41] *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3),* IETF (U. Blumenthal, B. Wijnen), Internet Proposed Standard, RFC 2574, April 1999.

[42] *IPSec Monitoring MIB*, IETF Draft (T. Jenkins, J. Shriver), http://www.ietf.org/internet-drafts/draft-ietf-ipsec-monitor-mib-01.txt, June, 1999.

[43] *DOMAIN NAMES – Implementation and Specification*, IETF (P. Mockapetris), Internet Proposed Standard, RFC 1035, November 1987.

[44] *PF_KEY Key Management API*, Version 2, IETF (D. McDonald, C. Metz, B. Phan), Internet Proposed Standard, RFC 2367, July 1998.

[45] *PKCS #9: Selected Attribute Types. Version 1.1*, RSA Laboratories, November, 1993.

[46] *PacketCable Electronic Surveillance Specification*, PKT-SP-PCES-I01-991230. December 30, 1999. Cable Television Laboratories, Inc. http://www.PacketCable.com./

[47] *FIPS-81, Federal Information Processing Standards Publication (FIPS PUB) 81, DES Modes of Operation, December 1980.*

[48] *RSVP Cryptographic Authentication*, IETF Draft (F. Baker, B. Lindell and M. Talwar), Internet Proposed Standard, RFC 2747, January 2000.

[49] *PacketCable CMS to CMS Signaling Specification*, PKT-SP-CMSS-D01-001010, October 10, 2000. Cable Television Laboratories, Inc. http://www.PacketCable.com./

[50] *Kerberos Set/Change Password: Version 2, IETF Draft (M. Swift, J.Trostle, J. Brezak, B. Gossman), http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-set-passwd-03.txt, April 2000.*

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

[52] *Internet X.509 Public Key Infrastructure Certificate and CRL Profile, IETF (Housley, R., Ford, W., Polk, W. and Solo, D.) Internet Proposed Standard, RFC2459, January 1999.*

2.2 Informative References

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

[2] *Randomness Recommendations for Security*, IETF Draft (Donald Eastlake, Stephen Crocker and Jeff Schiller) Internet Proposed Standard, RFC 1750, 1994.

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

3 TERMS AND DEFINITIONS

This document defines 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".
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.
Asymmetric Key	An encryption key or a decryption key used in a 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 the 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.
Confidentiality	A way to ensure that information is not disclosed to any one 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 head-end toward the subscriber location.
Encipherment	A method used to translate information in plaintext into ciphertext.
Encryption	A method used to translate information in plaintext into ciphertext.
Encryption Key	The key used in a cryptographic algorithm to translate the plaintext to ciphertext.
Endpoint	A Terminal, Gateway or MCU
EO	End Office

Errored Second	Any 1-sec interval containing at least one bit error.
ESP	IPSec Encapsulation Security Payload protocol that provides both IP packet encryption and optional message integrity, not covering the IP packet header.
ETSI	European Telecommunications Standards Institute
Event Message	Message capturing a single portion of a connection
FGD	Feature Group D signaling
F-link	F-Links are SS7 links that directly connect two SS7 end points, such as two SSPs. 'F' stands for "Fully Associated"
Flow [IP Flow]	A unidirectional sequence of packets identified by ISO 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.
Flow [DOCSIS Flow]	(a.k.a. DOCSIS-QoS "service flow"). A unidirectional sequence of packets associated with a SID and a QoS. Multiple multimedia streams may be carried in a single DOCSIS Flow.
FQDN	Fully Qualified Domain Name. Refer to IETF RFC 821 for details.
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 tot he edge of the PacketCable network.
H.323	An ISO standard for transmitting and controlling audio and video information. The H.323 standard requires the use of the H.225/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.
Keying Material	A set of cryptographic keys and their associated parameters, normally associated with a particular run of 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.
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.
PKINIT	The extension to the Kerberos protocol that provides a method for using public key cryptography during initial authentication.
Plaintext	The original (unencrypted) state of a message or data.
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 there by eliminating a host from having to support the services themselves.
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 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 users private key is kept secret and is the only key that can decrypt messages sent encrypted by the users 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 that it 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 by using encryption.
Subflow	A unidirectional flow of IP packets characterized by a single source and destination IP address and 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 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 (MF, R2, 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 head-end.
X.509 certificate	a public key certificate specification developed as part of the ITU-T X.500 standards directory

4 ABBREVIATIONS

This document uses the following abbreviations.

AAA	Authentication, Authorization and Accounting
AF	Assured Forwarding. A DiffServ Per Hop Behavior.
AH	Authentication header is 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 – an application specific field in the IPSec header that along with the destination IP address provides a unique number for each SA.
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, another way of saying AMA
BPI+	Baseline Privacy Interface Plus is the security portion of the DOCSIS 1.1 standard that runs on the MAC layer.
CBC	Cipher block chaining mode is 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.
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. In this specification "Call Agent" is part of the CMS that maintains the communication state, and controls the line side of the communication.
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.
CMTS	Cable Modem Termination System, the device at a cable head-end which implements the DOCSIS RFI MAC protocol and connects to CMs over an HFC network.
Codec	COder-DECoder
COPS	Common Open Policy Service Protocol is currently an internet draft, which describes a client/server model for supporting policy control over QoS Signaling Protocols and provisioned QoS resource management.
CoS	Class of Service. The type 4 tuple of a DOCSIS 1.0 configuration file.
CSR	Customer Service Representative
DA	Directory Assistance
DE	Default. A DiffServ Per Hop Behavior.
DHCP	Dynamic Host Configuration Protocol.
DHCP-D	DHCP Default – Network Provider DHCP Server
DNS	Domain Name Server
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. See Appendix B.
DOCSIS	Data Over Cable System Interface Specification.
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, i.e. assigned on the fly for each communication depending on the QoS requested
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 Encapsulation Security Payload protocol that provides both IP packet encryption and optional message integrity, not covering the IP packet header.
ETSI	European Telecommunications Standards Institute
FGD	Feature Group D signaling
F-link	F-Links are SS7 links that directly connect two SS7 end points, such as two SSPs. 'F' stands for "Fully Associated"
FQDN	Fully Qualified Domain Name. Refer to IETF RFC 821 for details.
HFC	Hybrid Fiber/Coax(ial [cable]), HFC system is a broadband bi-directional shared media transmission system using fiber trunks between the head-end and the fiber nodes, and coaxial distribution from the fiber nodes to the customer locations.
H.GCP	A protocol for media gateway control being developed by ITU.
HMAC	Hashed Message Authentication Code – a message authentication algorithm, based on either SHA-1 or MD5 hash and defined in RFC 2104.
HTTP	Hyper Text 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 in the Internet.
IKE	Internet Key Exchange is 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.
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
ISUP	ISDN User Part is a protocol within the SS7 suite of protocols that is used for call signaling within an SS7 network.
ISTP	Internet Signaling Transport Protocol
ISTP – User	Any element, node, or software process that uses the ISTP stack for signaling communications.
ITU	International Telecommunication Union
IVR	Interactive Voice Response System
LATA	Local Access and Transport Area
LD	Long Distance
LIDB	Line Information Database, containing information on customers required for real-time access such as calling card personal identification numbers (PINs) for real-time validation
LLC	Logical Link Control, used here to mean 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
MD5	Message Digest 5 – a one-way hash algorithm that maps variable length plaintext into fixed length (16 byte) ciphertext.
MDCP	A media gateway control specification submitted to IETF by Lucent. Now called SCTP.
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.

MG	The media gateway provides the bearer circuit interfaces to the PSTN and transcodes the media stream.
MGC	A Media Gateway Controller is the overall controller function of the PSTN gateway. It receives, controls and mediates call-signaling information between the PacketCable and PSTN.
MGCP	Media Gateway Control Protocol. Protocol follow-on to SGCP.
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 MAC.
MMC	Multi-Point Mixing Controller. A conferencing device for mixing media streams of multiple connections.
MSO	Multi-System Operator, a cable company that operates many head-end locations in several cities.
MSU	Message Signal Unit
MTA	Media 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 is a set of two protocols (MTP 2, 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; the layer that 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 that defines the analog color television, broadcast standard used today in North America.
OSP	Operator Service Provider
OSS-D	OSS Default – Network Provider Provisioning Server
OSS	Operations Systems Support. The back office software used for configuration, performance, fault, accounting and security management.
PAL	Phase Alternate Line – the European color television format that evolved from the American NTSC standard.
PDU	Protocol Data Unit
PKCS	Public Key Cryptography Standards, published by RSA Data Security Inc. Describes 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.
PKCROSS	Utilizes PKINIT for establishing the inter-realm keys and associated inter-realm policies to be applied in issuing cross realm service tickets between realms and domains in support of Intradomain and Interdomain CMS-to-CMS signaling (CMSS).
PHS	Payload Header Suppression, a DOCSIS technique for compressing the Ethernet, IP and UDP headers of RTP packets.
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.
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.
QCIF	Quarter Common Intermediate Format
QoS	Quality of Service, guarantees network bandwidth and availability for applications.

RADIUS	Remote Access Dial-In User Service, an internet protocol (RFC 2138 and RFC 2139) 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.
RC4	A variable key length stream cipher offered in the ciphersuite, used to encrypt the media traffic in PacketCable.
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	Standard 4-pin modular connector commonly used in the United States for connecting a phone unit into the wall jack
RKS	Record Keeping Server, the device which collects and correlates the various Event Messages
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 Protocol, a protocol defined in RFC 1889 for encapsulating encoded voice and video streams.
S-MTA	Standalone MTA – a single node that contains an MTA and a non DOCSIS MAC (e.g. ethernet).
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 BPI+ security protocol, part of the DOCSIS 1.1 specification.
SCCP	The Signaling Connection Control Part is a protocol within the SS7 suite of protocols that provides two functions in addition to those that are provided within MTP. The first is the ability to address applications within a signaling point. The second function is Global Title Translation.

SCP	A Service Control Point is a Signaling Point within the SS7 network, identifiable by a Destination Point Code that provides database services to the network.
SCTP	Simple Control Transmission Protocol.
SDP	Session Description Protocol.
SDU	Service Data Unit. Information that is 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. Any 32-bit SFID must not conflict with a zero-extended 14-bit SID. SFIDs are considered to be in either the upstream direction (USFID) or downstream direction (DSFID). USFIDs and DSFIDs 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 is an application layer control (signaling) protocol for creating, modifying and terminating sessions with one or more participants.
SIP+	Session Initiation Protocol Plus is an extension to SIP.
SNMP	Simple Network Management Protocol
SOHO	Small Office/Home Office
SS7	Signaling System Number 7. SS7 is an architecture and set of protocols for performing out-of-band call signaling with a telephone network.

SSP	Signal 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. An STP is a node within an SS7 network that routes signaling messages based on their destination address. It 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 used to grant Kerberos tickets.
TGW	Telephony Gateway
TIPHON	Telecommunications & Internet Protocol Harmonization Over Network.
TLV	Type-Length-Value 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

WBEM	Web-Based Enterprise Management (WBEM) is the umbrella under which the DMTF (Desktop Management Task Force) will fit its current and future specifications. The goal of the WBEM initiative is to further management standards using Internet technology in a manner that provides for interoperable management of the Enterprise. There is one DMTF standard today within WBEM and that is CIM (Common Information Model). WBEM compliance means adhering to the CIM. See www.dmtf.org

5 ARCHITECTURAL OVERVIEW OF PACKETCABLE SECURITY

5.1 PacketCable Reference Architecture

Security requirements had been defined for every signaling and media link within the PacketCable IP network. Thus, 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 1.2 architecture. For a more detailed specification, refer to the PacketCable Architecture Technical Report [1] and the PacketCable 1.2 Architecture Technical Report [2].

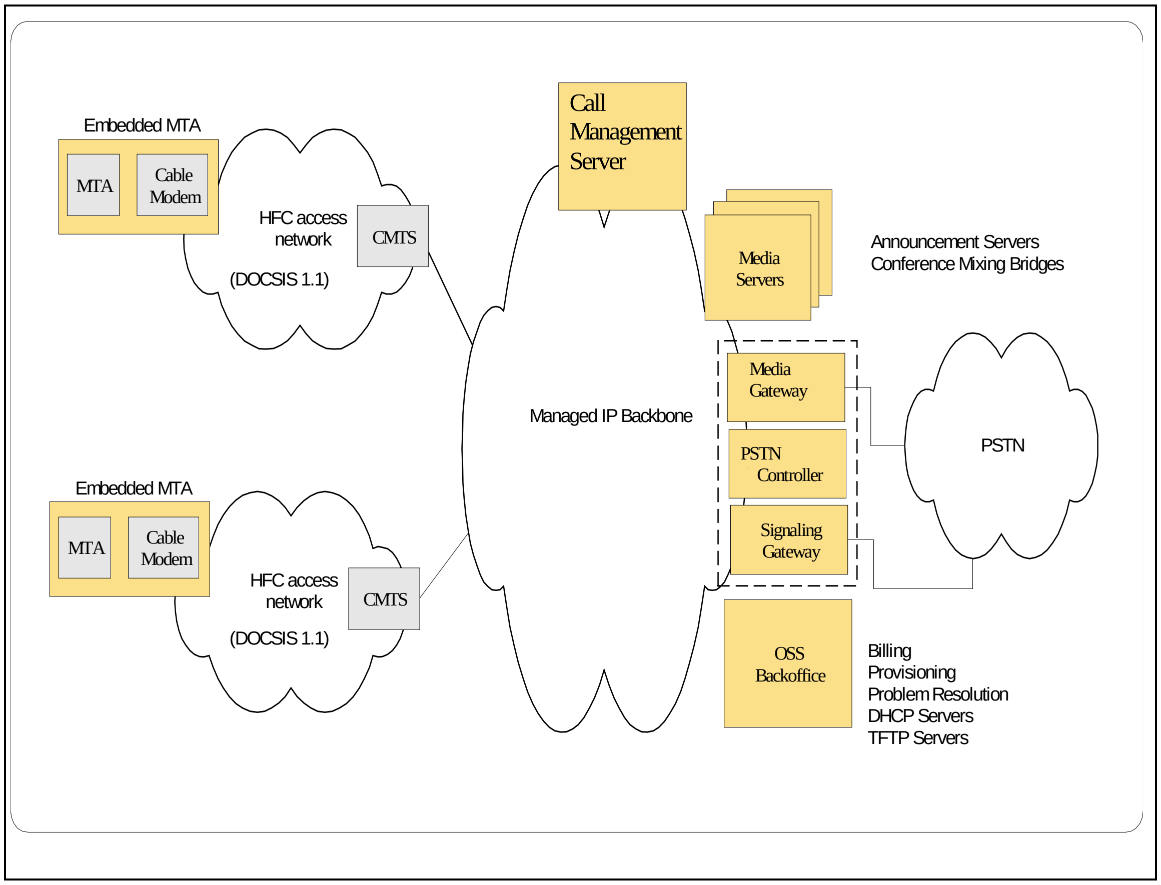

Figure 1. PacketCable Single Zone Architecture

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 [13] and MAC layer protocols. DOCSIS BPI+ [14] and QoS protocols are enabled over this link.

There is signaling between the MTA and the CMTS, for the purpose of Dynamic QoS [4]. This includes both custom UDP messages defined specifically for PacketCable

DQoS, as well as standard RSVP messages, which run directly over the IP layer in the protocol stack. PacketCable also provides guaranteed QoS for each voice communication between domains with Interdomain QoS (IQoS)[5].

Since BPI+ key management and privacy runs between the Cable Modem and the CMTS, it does not help in authenticating the identity of the MTA. Therefore, all protocols that run between the MTA and the CMTS have additional security requirements that cannot be met by BPI+. Also, since the MTA communicates with other devices on the PacketCable beyond the CMTS, these communications must be protected as well.

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 Dynamic Quality of Service (DQoS) establishment. The CMS also performs queries at the PSTN Gateway for LNP (Local Number Portability) and other services necessary for voice communications, including interface 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 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 with the detailed specifications given only to the security parameters. For a full specification on MTA provisioning and customer enrollment, refer to [7].

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)[4].

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 layer 2 signaling or DOCSIS 1.l QoS between the MTA and the CMTS on both sides of the connection. QoS reservations are also forwarded to the IP Backbone between the CMTSs, but the specifications of the Backbone reservations are currently out of the scope of PacketCable. Therefore, the corresponding security specifications are also out of scope.

5.1.3.3 Interdomain Quality of Service

PacketCable provides guaranteed QoS for each voice communication between domains with Interdomain QoS (IQoS)[5]. DiffServ allows IP traffic to be marked with different DiffServ Code Points (DSCP) to obtain different queuing treatment on routers. Different queuing treatments in each router are called per hop behavior (PHB) that is a mechanism for enforcing QoS for different flows in the IP Backbone.

5.1.3.4 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.

5.1.3.5 Call Signaling

The call signaling architecture defined within PacketCable is Network Call Signaling (NCS). The CMS is used to control call setup, termination and most other call signaling functions. In the NCS architecture[3], the Call Agent function within the CMS is used in call signaling, and it utilizes the MGCP protocol.

5.1.3.6 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 [39] and [40].

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 both security requirements and specifications for the PSTN Gateway links.

When communications from an MTA to a PSTN phone is made, bearer channel traffic is passed directly between an MTA and a MG. The protocols used in this case are RTP and RTCP, the same 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 scope of this document.

5.1.3.7 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

[1] In general, it is possible for an MTA-to-MTA or MTA-to-PSTN connection to cross 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 an 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.

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 represent 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, etc. 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 [10].

5.1.3.8 Audio Server Services

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

5.1.3.8.1 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.

5.1.3.8.2 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.

5.1.3.9 Electronic Surveillance

The event interface between the CMS and the DF provides descriptions of calls, 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.

[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..

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

The diagram below contains the interfaces that were analyzed for security.

There are additional interfaces that are 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 diagram below.

The interfaces for which security is not required in PacketCable are not listed in Figure 2 below.

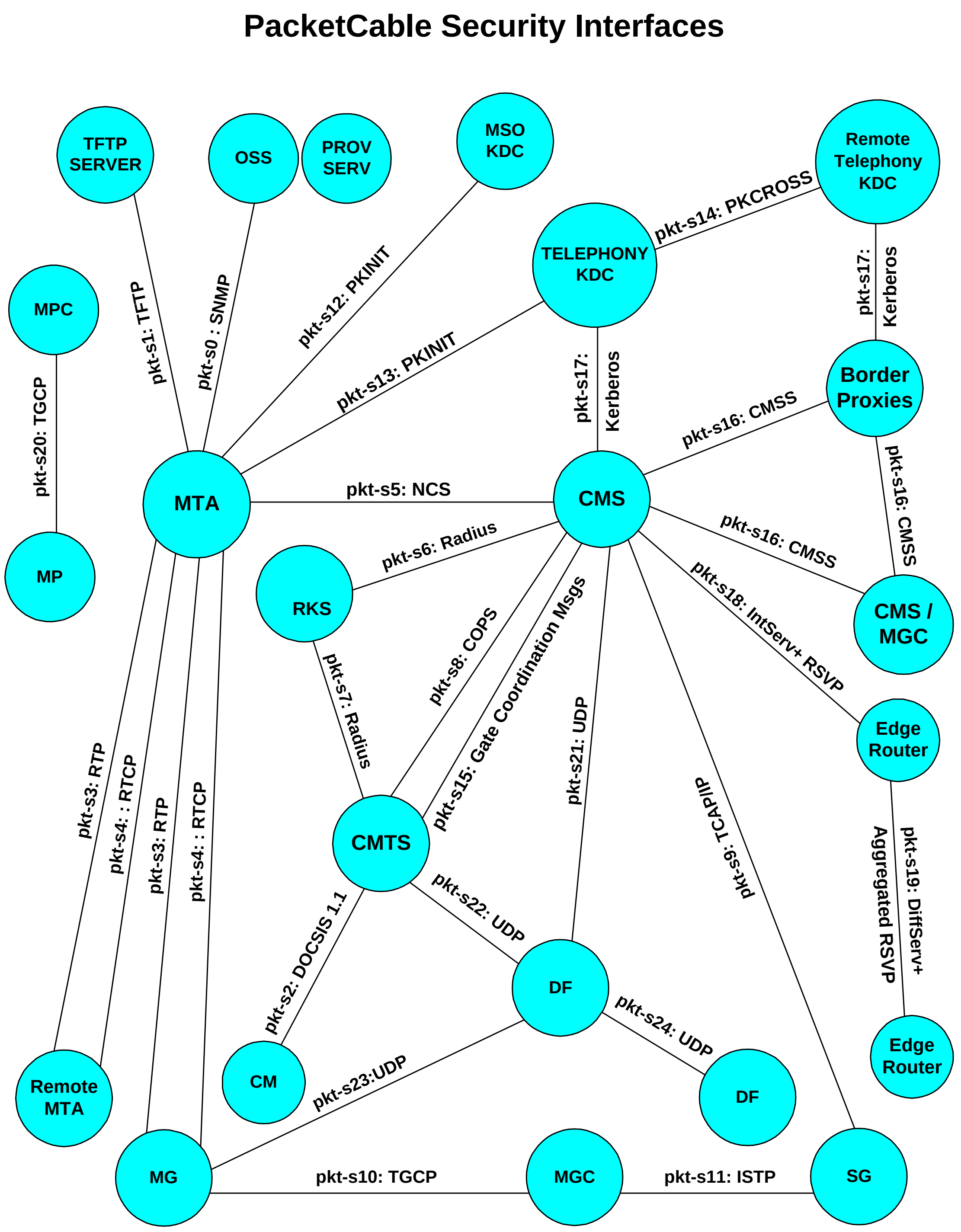

Figure 2: PacketCable Secured Interfaces

Below 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 each vendor and each MSO 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 by someone other than the owner, 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 particular 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 – 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 way to revoke 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 the 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. (Note that 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 the Bearer Channel Information threats. These attacks were 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 completely inapplicable. An owner of an MTA may distribute clones to the 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 with 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 customer's 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 the 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 that is 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 were already described under the Service Theft threats.

MTA cloning attacks mounted by the actual owner of the MTA can be theoretically 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 masquerade (using the above mentioned cloning and protocol manipulation techniques) is common or is easily achievable, a customer may repudiate a particular communication (and, e.g., 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.

- **Primary Threats**
    - **Theft of Service**. Attacks are:
        - **Subscription Fraud**. This attack is prevalent 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 a 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 the ones for bearer channel privacy and are not described here:
    - **Simple snooping**
    - **MTA clones**
    - **Protocol manipulation**
    - **Off-line cryptanalysis**
    - **Service disruption**

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.

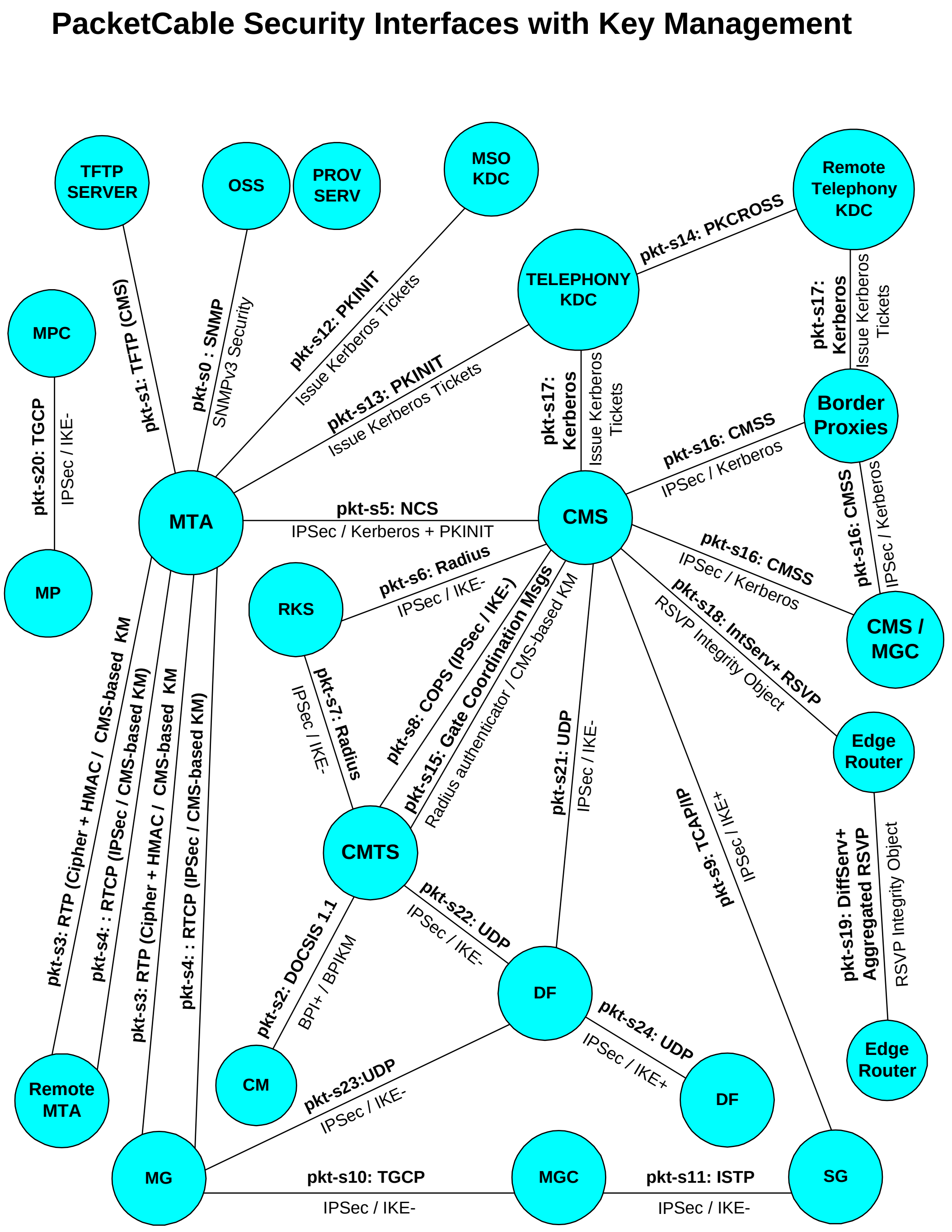

Figure 3. PacketCable Security Interfaces with Key Management

In the above diagram, each interface label looks like:

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

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

The following abbreviations were used in the above diagram:

IKE-	IKE with pre-shared keys
IKE+	IKE with X.509 certificates
CMS-based KM	Keys randomly generated and exchanged inside signaling messages

Here is a table that 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	SNMPv3 : The initial SNMPv3 INFORM from the MTA to the Provisioning Server, followed by optional SNMP GET(s) by the SNMP Manager are used to query MTA device capabilities. This occurs at the time where SNMPv3 keys may not be available, and security is provided with an RSA signature, formatted according to Cryptographic Message Syntax. Later, standard SNMPv3 security is enabled to the OSS.
pkt-s1	MTA - TFTP	TFTP: MTA Configuration file download. The MTA downloads a secure configuration file (with TFTP-get) that is signed by the TFTP server and sealed with the MTA public key, with a Cryptographic Message Syntax wrapper.
pkt-s2	CM - CMTS	DOCSIS 1.1: Secured with BPI+ using BPI Key Management. BPI+ privacy layer 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 HMAC (Hashed Message Authentication Code). Keys are randomly generated and exchanged by the two endpoints inside the signaling messages via the CMS. .
pkt-s4	MTA – MTA MTA - MG	RTCP: RTCP control protocol for RTP. Message integrity and encryption provided with IPSec. Key management is same as for RTP, keys are randomly generated and exchanged by the two endpoints inside the signaling messages via the CMS..
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: Radius billing events sent by the CMS to the RKS. Radius authentication keys are hardcoded to 0. IPSec is used for message integrity, as well as privacy. Key management is IKE-.
pkt-s7	RKS – CMTS	Radius: Radius events sent by the CMTS to the RKS. Radius authentication keys are hardcoded to 0. IPSec is used for message integrity, as well as privacy. Key management is IKE-.
pkt-s8	CMS - CMTS	COPS: COPS protocol between the GC and the CMTS, used to download QoS authorization to the CMTS. Security is provided with IPSec for message integrity as well as privacy. Key Management is IKE-.

Interface	Components	Description
pkt-s9	CMS - SG	TCAP/IP: CMS queries the PSTN Gateway for LNP (Local Number Portability) and other voice communications services. Security is provided with IPSec for message integrity as well as privacy. Key Management is IKE-.
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-.
pkt-s11	MGC - SG	ISTP: PacketCable interface to the PSTN Signaling Gateway. IPSec is used for both message integrity and privacy. Key management is IKE-.
pkt-s12	MTA – MSO KDC	PKINIT: An AS-REQ message is sent to the KDC as before, except public key cryptography is used in the initial authentication step. The KDC verifies the certificate and issues a ticket granting ticket (TGT). The KDC authenticates the message using its public key signature.
pkt-s13	MTA – Telephony KDC	PKINIT: See pkt-s12 above.
pkt-s14	Telephony KDC – Remote Telephony KDC	PKCROSS utilizes PKINIT for establishing the inter-realm keys and associated inter-realm policies to be applied in issuing cross realm service tickets between realms and domains in support of Intradomain and Interdomain CMS-to-CMS signaling (CMSS).
pkt-s15	CMS – CMTS	Gate Coordination messages for DQoS. Message integrity is provided with an application-layer (Radius) authenticator. Keys are distributed by local CMS over COPS.
pkt-s16	CMS – CMS CMS – EBP	SIP: IPSec is used for both message integrity and privacy. Key management is Kerberos.
pkt-s17	CMS – Telephony KDC EBP – Remote Telephony KDC	Kerberos: PKINIT requests (AS Request – AS Reply) for a TGT between Kerberos realms, both Intradomain and Interdomain. The TGT request generates a cross-realm TGT request using the TGS Request – TGS Reply and the PKCROSS mechanisms. The cross-realm TGS Reply is used to generate the AS Request – AS Reply needed to establish Security Associations between two domains.
pkt-s18	CMS - ER	IntServ + RSVP: Secured by using RSVP Integrity Objects
pkt-s19	ER - ER	Aggregated RSVP: Secure by using RSVP Integrity Objects
pkt-s20	MPC - MP	TGCP: IPSec is used for both message integrity and privacy. Key management is IKE-.
pkt-s21	DF - CMS	UDP: IPSec is used for both message integrity and privacy. Key management is IKE-.
pkt-s22	DF - CMTS	UDP: IPSec is used for both message integrity and privacy. Key management is IKE-.
pkt-s23	DF - MG	UDP: IPSec is used for both message integrity and privacy. Key management is IKE-.
pkt-s24	DF - DF	UDP: IPSec is used for both message integrity and privacy. Key management is IKE+.

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 a great expense.

Most messages secured in this specification either transit 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 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 not 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 that is 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 Protection From Being Compromised

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 possibility of this happening is very low. Low enough so 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 of 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 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 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 were 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. And, 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
- 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. And, 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 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

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

    - Downloading invalid or wrong billing information to the CMTS
    - Combination of both of the above
- Loss of privacy, since the CMS distributes a bearer channel keys
- Unauthorized allocation of QoS
- Unauthorized disclosure of customer identities, 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 identities, 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 MTA completely inoperable.
    - Degrade voice communications performance on the PacketCable network or on the HFC network.
    - Enable 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 identities, location (e.g. IP address), usage patterns and a list of services to which the customer subscribes

6 SECURITY MECHANISMS

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 [29].

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 [30] 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 [29].

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 [31]. Within PacketCable, the same Transform Identifiers are used by all IPSec key management protocols: IKE, Kerberos and application layer (embedded in IP signaling messages).

For PacketCable, the following IPSec Transform Identifiers are supported (all of which are specified in [32]):

Table 2. IPSEC ESP Transform Identifiers

Transform ID	Value (Hex)	Key Size (in bits)	MUST Support	Description
ESP_3DES	3	192	yes	3-DES in CBC mode.
ESP_RC5	4	128	no	RC5 in CBC mode
ESP_IDEA	5	128	no	IDEA in CBC mode
ESP_CAST	6	128	no	CAST in CBC mode
ESP_BLOWFISH	7	128	no	BLOWFISH in CBC mode
ESP_NULL	11	0	yes	Encryption turned off

The ESP_3DES and ESP_NULL Transform Ids MUST be supported. For all of the above transforms, the CBC Initialization Vector (IV) is carried in the clear inside each ESP packet payload[32].

IKE allows the 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.

Security specification for each PacketCable interface lists whether or not encryption (confidentiality) is required. On each interface that requires confidentiality, the ESP_NULL transform MUST NOT be used.

6.1.2.2 IPSec ESP Authentication Algorithms

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 [31]. Within PacketCable, the same Authentication Algorithms are used by all IPSec key management protocols: IKE, Kerberos and application layer (embedded in IP signaling messages).

For PacketCable, the following IPSec Authentication Algorithms are supported (all of which are specified in [32]):

Table 3. IPSEC Authentication Algorithms

Authentication Algorithm	Value (Hex)	Key Size (in bits)	MUST Support	Description
HMAC-MD5	1	128	yes (also required by [31])	MD5 HMAC
HMAC-SHA	2	160	yes	SHA-1 HMAC

Security specification for each PacketCable interface lists whether or not message integrity is required. The HMAC-MD5 and HMAC-SHA authentication algorithms MUST be supported.

6.1.2.3 Replay Protection

In general, IPSec provides an optional service of replay protection (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 anti-replay service is turned on, an IPSec sequence number cannot overflow and roll over to 0. Before that happens, a new Security Parameter must be created, as specified in [30].

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

Because within PacketCable, IPSec is used on a number of different interfaces with different security and performance requirements, 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 6.4.3.1), and in some cases IPSec keys are distributed over protected signaling interfaces (see section 7.6.2.2).

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. 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 one is about to expire or there isn't one on a particular interface).

In addition, some network elements are required to run multiple key management protocols. In particular, the 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 MTA MUST get its IPSEC keys for RTCP packets from NCS signaling messages.

The PF_KEY interface (see [44]) 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 would signal the event to all registered key management applications. Based on the Identity Extension associated with that Security Association, each key management application would decide 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 [34]. It is utilized on the interfaces, where:

- There is not a very large number of connections (on the order of 100,000 or above)
- 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 Parameter is unexpectedly lost by one of the endpoints.

IKE is a peer-to-peer key management protocol. It consists of 2 phases. In the 1st phase, a shared secret is negotiated via a Diffie-Hellman key exchange. It is then used to authenticate the 2nd IKE phase. The 2nd IKE phase negotiates another secret, used to derive keys for the IPSec ESP protocol.

6.2.2 PacketCable Profile for IKE

6.2.2.1 1st IKE Phase

There are multiple modes defined for authentication during the 1st 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 [34]).

6.2.2.1.2 IKE Authentication with Public Key Encryption

PacketCable MUST NOT utilize this IKE authentication mode. 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.

6.2.2.2 2nd IKE Phase

In the 2nd IKE phase, an IPSec ESP SA is established, including the IPSec ESP keys and ciphersuites. It is possible to establish multiple Security Parameters with a single 2nd phase IKE exchange.

First, a shared 2nd phase secret is established, and then all the IPSec keying material is derived from it using a one-way function, specified in [34].

The 2nd 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 2nd IKE phase (in order to avoid the associated performance penalties).

The 2nd IKE phase is authenticated using a shared secret that was established in the 1st phase. Supported authentication algorithms are the same as the ones specified for IPSec in section 6.1.2.2.

6.2.2.3 Encryption Algorithms for IKE Exchanges

Both phase 1 and 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 [34]. The use of Diffie-Hellman groups within PacketCable is exactly how it is specified in [34].

6.3 SNMPv3

All SNMP-based network management within PacketCable MUST run over SNMPv3 with security specified by [41]. 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 6.5.7). In addition, SNMPv3 key management techniques specified in [41] 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	64	yes	DES in CBC mode.
SNMPv3_NULL	0x20	0	yes	Encryption turned off

The DES encryption transform for SNMPv3 is specified in **[41]**. 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 (all of which are specified in **[41]**):

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 [41])	MD5 HMAC
SNMPv3_HMAC-SHA-1	0x22	160	No (SHOULD be supported)	SHA-1 HMAC

6.4 Kerberos / PKINIT

6.4.1 Definitions

- Application Server: In this section, the generic term Application Server means any Kerberized server (e.g. CMS, Provisioning Server).
- PKINIT: Kerberos Public Key INITial authentication extension.

6.4.2 Overview

PacketCable utilizes the concept of Kerberized IPSec for signaling between the CMS and MTA. That is, the ability to create IPSec security associations using keys derived from the session key of a Kerberos ticket. On this interface, Kerberos [18] is utilized with the PKINIT public key extension (also see Appendix B).

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 3 phases of Kerberos-based key management for IPSec:

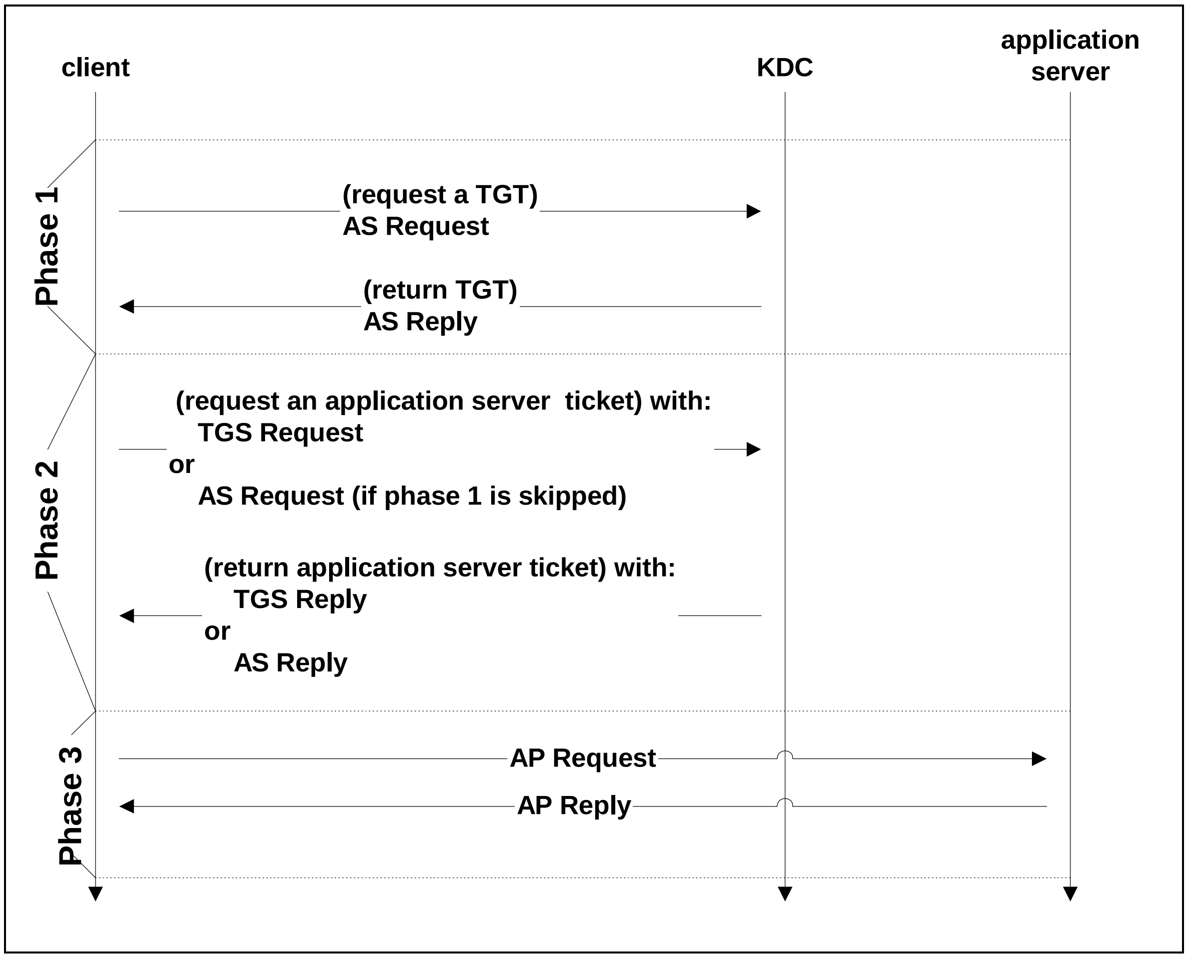

Figure 4. Kerberos-Based Key Management for IPSEC

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 6.4.3.2). 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 6.4.3). Both the TGT and the Application Server tickets used within PacketCable have a relatively long lifetime (days or weeks), which is acceptable as 3-DES, a reasonably strong symmetric cryptography, 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 some 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, where a server (e.g. CMS) does not need to maintain a SA for every client (e.g. MTA) that it controls. It also is possible that a group of 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 a different profile for SNMPv3 vs. 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.

6.4.3 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 would plug in is specified in [17].

In this section, the MTA is 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 the subsequent sections of this document.

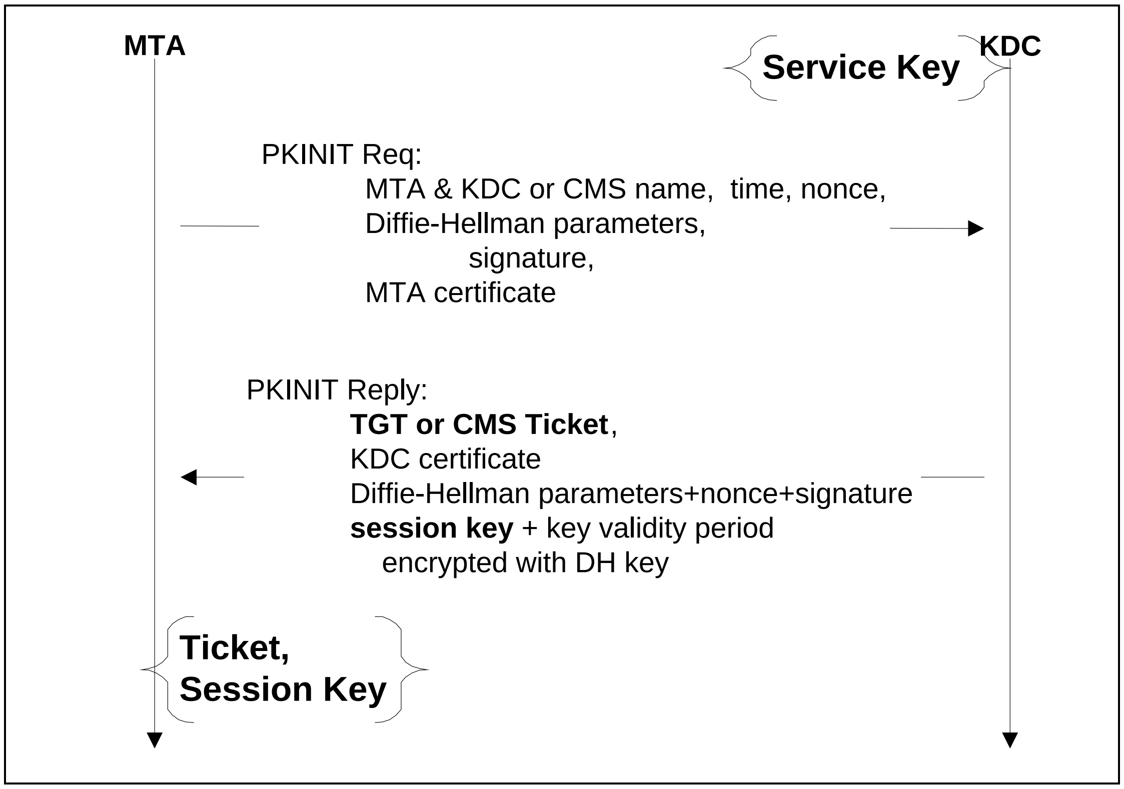

Figure 5. PKINIT Exchange

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 [18]). Its format is based on the MTA's X.500 name in the certificate, as specified in Appendix B.
    - KDC or Application Server (Kerberos principal) name – found in the KDC-REQ-BODY Kerberos structure (see [18]). For the format used in PacketCable, see section 6.4.6.3.
    - Time – found in the PKAuthenticator structure, specified by PKINIT (Appendix B).
    - Nonce - found in the PKAuthenticator structure, specified by PKINIT (Appendix B). There is also a 2nd nonce in the KDC-REQ-BODY Kerberos structure.
    - Diffie-Hellman parameters, signature and MTA certificate – these are all specified by PKINIT (Appendix B) and their use in PacketCable is specified in section 6.4.3.1.1.
- PKINIT Reply

- TGT or Application Server Ticket – found in the KDC-REP Kerberos structure (see [18]).
- KDC Certificate, Diffie-Hellman parameters, signature – these are all specified by PKINIT (Appendix B) and their use in PacketCable is specified in section 6.4.3.1.2.
- Nonce – found in the KdcDHKeyInfo structure, specified by PKINIT (Appendix B). 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 [18]). 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 [18]).

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 server is either a Application Server or a KDC (in which case the ticket is the TGT).

6.4.3.1 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 **$\textbf{Ticket}_{\textbf{EXP}}$** and on the PKINIT Grace Period **$\textbf{PKINIT}_{\textbf{GP}}$**. The MTA MUST initiate the PKINIT exchange at the time: **$\textbf{Ticket}_{\textbf{EXP}}$ - $\textbf{PKINIT}_{\textbf{GP}}$**. On the interfaces where $\text{PKINIT}_{\text{GP}}$ is not defined, the MTA SHOULD perform PKINIT exchanges on-demand.

In the case where PKINIT is used to obtain a Application Server ticket directly, the use of the grace period accounts for a possible clock skew between the MTA and the Application Server. If the MTA is late with the PKINIT exchange, it still has until **$\textbf{Ticket}_{\textbf{EXP}}$** 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 Application Server. Synchronizing the PKINIT exchange with the AP Request/Reply exchange is not required as long as the AP Request/AP Reply exchange is 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.

Once an MTA receives an AS Reply (with the PKINIT Reply in it), 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 (which is usually the case with existing Kerberos implementations). Thus, the MTA will be able to re-use the same Kerberos ticket after a reboot, avoiding the need to perform PKINIT again, with the associated overhead of public key operations.

Since an MTA is not required to save the ticket, the MTAs that don't follow the above recommendation should not adversely affect the performance of call signaling. Therefore, 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 the Application Server 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 $\mathbf{PKINIT_{GP}}$ MUST be at least 15 minutes.

6.4.3.1.1 PKINIT Request

The PKINIT Request message (PA-PK-AS-REQ) in Appendix B 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, see [19], 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.)

  The PKAuthenticator data type is specified by PKINIT.
- **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 chapter 8.
- **crls –** MUST NOT be filled in by the MTA.
- **signerInfos** – MUST be a sequence 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 and the current required signature algorithm MUST be RSA over a SHA-1 digest. A PacketCable client MUST NOT insert any optional fields in this data structure. The signature algorithm MUST be RSA over a SHA-1 digest.

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. For details, refer to Appendix B. PacketCable requirements for the Diffie-Hellman parameters (i.e., prime and generator) MUST follow the IKE specification in [34].

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.

6.4.3.1.2 PKINIT Reply

The PKINIT Reply message (PA-PK-AS-REP) in Appendix B 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 [17]. The AS Reply MUST be encrypted with 3-DES CBC, where the corresponding Kerberos **etype** value MUST be **des3-cbc-md5**. 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**.

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 6.4.3.1.1):

- **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 {
   nonce [0] INTEGER,
   subjectPublicKey [2] BIT STRING
  }
  ```

  Where the **nonce** must be the same nonce that was passed in by the client in the PKINIT Request and **subjectPublicKey** is the Diffie-Hellman public value generated by the KDC. The Diffie-Hellman-derived key is used to directly encrypt part of the **AS Reply**.

  - **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 Certificate MUST also be present. The Service Provider Certificate MUST also be present in this field. If 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.
  - **crls –** this optional field MAY be filled in by the KDC.
  - **signerInfos –** MUST be a sequence 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. PacketCable MUST NOT use all optional fields in this data structure and the signature algorithm for now MUST be RSA over a SHA-1 digest. A PacketCable KDC MUST NOT insert any optional fields in this data structure. The signature algorithm MUST be RSA over a SHA-1 digest.

6.4.3.1.2.1 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 the **KRB-ERROR,** as defined in [18]. Any error code that is defined in Appendix B 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 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.

6.4.3.1.2.1.1 Clock Skew Error

When the KDC clock and the client clock are off by more than the limit for a clock skew (usually 5 minutes), an error code KRB_AP_ERR_SKEW MUST be returned. 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 use it to adjust its clock in subsequent PKINIT and AP Request messages.

In the case that a PKINIT request failed due to a clock skew error, an MTA MUST immediately retry after adjusting its clock.

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, where a rogue KDC attempts to fool an MTA into accepting an expired KDC certificate.

6.4.3.2 Profile for the Kerberos AS Request / AS Reply Messages

As mentioned earlier, the PKINIT Request and Reply are really pre-authenticator fields, embedded into the AS Request / AS Reply messages. Other pre-authenticators MUST NOT be used in combination with PKINIT.

The client’s Kerberos principal name and realm in these messages MUST be formed based on the client’s X.500 name and on the client certificate issuer’s name respectively, as specified in Appendix B.

The optional fields **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. None of the Kerberos ticket flags are currently supported within PacketCable.

In the AS Reply, **key-expiration**, **starttime**, **renew-till** and **caddr** optional fields MUST NOT be present.

The encrypted part of the AS Reply MUST be encrypted with the encryption type set to **des3-cbc-md5**. The following data MUST be concatenated and processed in the following sequence before being encrypted with 3-DES CBC, IV=0:

(1) 8-byte random byte sequence, called a **confounder**

(2) And MD5 checksum, calculated as specified in [18]

(3) AS Reply part that is to be encrypted

(4) Random padding up to a multiple of 8

6.4.3.3 Profile for Kerberos Tickets

In Kerberos Tickets, **caddr**, **authorization-data**, **starttime** and **renew-till** optional fields MUST NOT be present. None of the Kerberos ticket flags are currently supported within PacketCable.

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 6.4.3.2.

6.4.4 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 6.4.3.2. All optional fields inside PA-ENC-TS-ENC MUST be present for PacketCable. The pachecksum field MUST be a keyed checksum of type des3-cbc-md5 and 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, 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 **pktcKdcToMtaMaxClockSkew** then KDC MUST reply with a clock skew error message and the MTA MUST respond to this error message as specified in section 6.4.3.1.2.1.1.

If the realm, target server name (e.g. the name of the TGS), 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 is returned. The KDC MUST remember any such preauthenticator presented within pktcKdcToMtaMaxClockSkew, so that a replay attempt is guaranteed to fail.

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

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

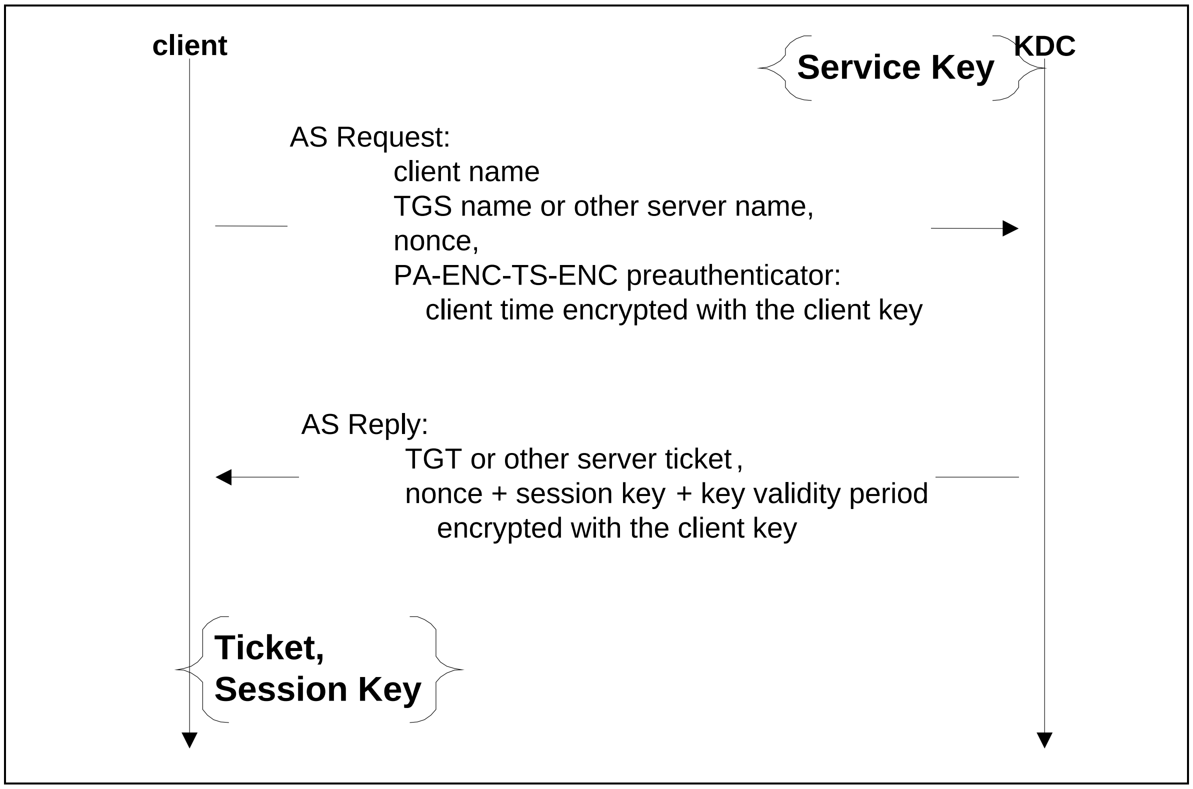
client
KDC
Service Key
AS Request:
client name
TGS name or other server name,
nonce,
PA-ENC-TS-ENC preauthenticator:
client time encrypted with the client key
AS Reply:
TGT or other server ticket,
nonce + session key + key validity period
encrypted with the client key
Ticket,
Session Key

Figure 6. Symmetric Key AS Request / AS Reply Exchange

6.4.4.1 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 6.4.3.2) 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 [18]. 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 a 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.

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).

6.4.5 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 [17].

A TGS Request includes an KRB_AP_REQ data structure (the same one that is used in an AP Request, see section 6.4.5.1). 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.

The diagram below illustrates the TGS Request / TGS Reply exchange:

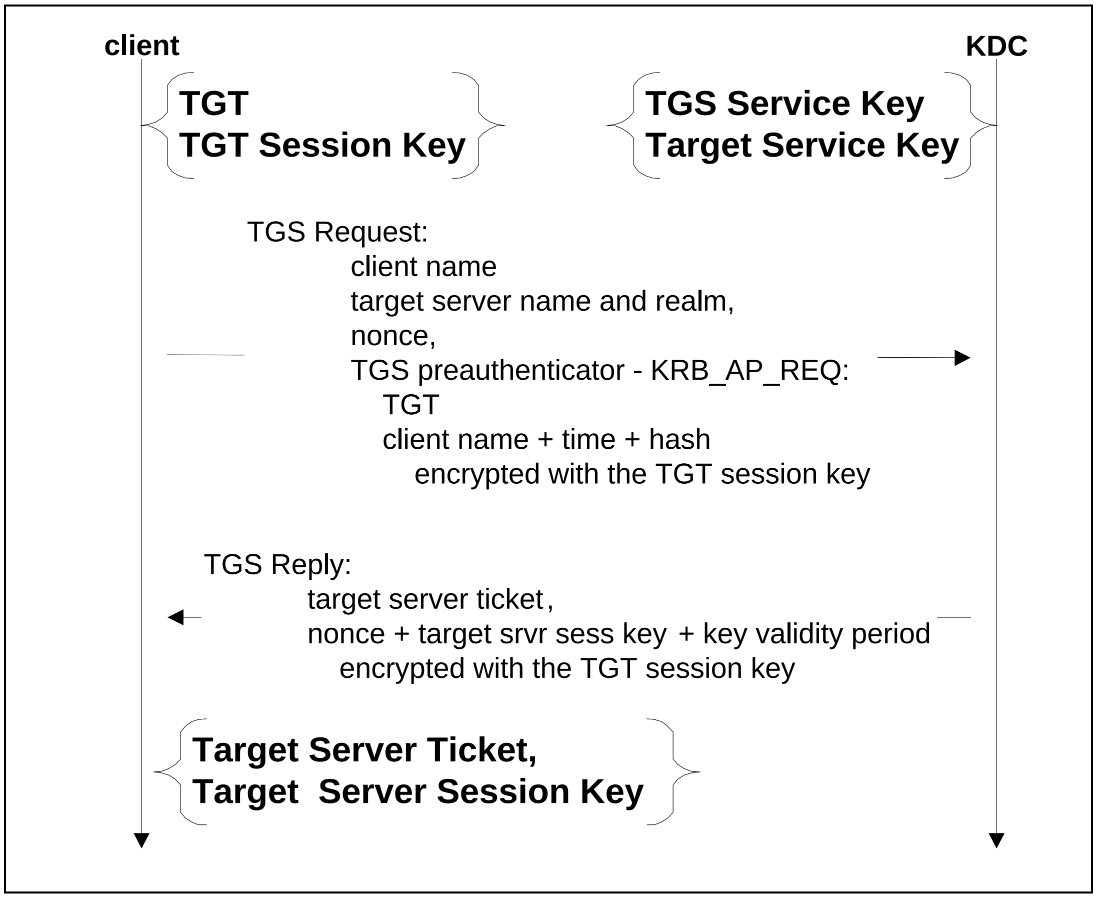
client
KDC
TGT
TGT Session Key
TGS Service Key
Target Service Key
TGS Request:
client name
target server name and realm,
nonce,
TGS preauthenticator - KRB_AP_REQ:
TGT
client name + time + hash
encrypted with the TGT session key
TGS Reply:
target server ticket,
nonce + target srvr sess key + key validity period
encrypted with the TGT session key
Target Server Ticket,
Target Server Session Key

Figure 7. Kerberos TGS Request / TGS Reply Exchange

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

- TGS Request
    - Client (principal) name, target server (principal) name and realm, nonce – found in the KDC-REQ-BODY Kerberos structure (see [18]).
    - TGS preauthenticator - found in the KDC-REQ Kerberos structure, inside the padata field (see [18]). 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 [18]).

- Target server session key, nonce, key validity period – found in the EncKDCRepPart Kerberos structure (see [18]).

In general, the TGS Request / Reply exchange may be performed on-demand - whenever an Application Server ticket is needed to establish Security Parameters. However, there may be PacketCable elements (e.g. MTAs) that are required to always possess a valid ticket for a particular Application Server (e.g. CMS) to improve efficiency in the load balancing scenarios. In those cases the client MUST initiate the TGS Request / Reply exchange at the time: $\mathbf{Ticket_{EXP} - TGS_{GP}}$. Here, $\mathbf{Ticket_{EXP}}$ is the expiration time of the current Application Server ticket and $\mathbf{TGS_{GP}}$ is the TGS Grace Period.

Once a client receives a TGS 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 TGS Request again.

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.

6.4.5.1 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:

- Options in the ap-options field MUST NOT be present
- The ticket is the TGT.
- 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.
- The authenticator MUST be encrypted using 3-DES CBC with the following Kerberos etype: **des3-cbc-md5**.

The optional fields **enc-authorization-data, additional-tickets** and **rtime** in the **KDC-REQ-BODY** MUST NOT be present in the TGS Request. All other optional fields in the TGS Request MAY be present for PacketCable. None of the Kerberos ticket flags are currently supported within PacketCable.

6.4.5.2 TGS Reply Profile

In the AS Reply, **key-expiration**, **starttime**, **renew-till** and **caddr** optional fields MUST NOT be present. The encrypted part of the AS Reply MUST be encrypted with the encryption type set to **des3-cbc-md5**, using the same procedure as described in section 6.4.3.2.

6.4.5.3 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 [18]. 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 **des3-cbc-md5**, calculated using the procedure described in section 6.4.3.2.

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

6.4.6 Kerberos Server Locations and Naming Conventions

6.4.6.1 Kerberos Realms

In PacketCable, a Kerberos Realm has a one-to-one correspondence with a zone. Each MTA MUST be configured with a single Kerberos realm name. A realm name MAY use the same syntax as a domain name. For a full specification of Kerberos realms, refer to [18].

6.4.6.2 KDC

Kerberos principal name for the local KDC 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 Appendix D. For example, let us say that realm ASDF.COM contains two KDCs: kdc1.asdf.com and kdc2.asdf.com. The DNS SRV records in this case are:

```
_kerberos._udp.ASDF.COM. IN     SRV   0 0 88 kdc1.asdf.com.
_kerberos._udp.ASDF.COM. IN     SRV   1 0 88 kdc2.asdf.com.
```

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

6.4.6.3 CMS

MTA configuration includes a list of CMS DNS names or IP addresses. In addition, each CMS entry MUST include a CMS Kerberos Principal Name, with the realm name not specified (since all CMS entries for one MTA are in the same realm).

Kerberos specification in general allows for principal names to contain an unlimited number of components, of the form <comp1>/<comp2>/<comp3>@<realm>. For PacketCable, each principal name MUST contain only a single component.

A Kerberos principal name for a CMS might be **CMS1@sandiego.company1.com**. In this example, the realm is **sandiego.company1.com** and it will be a separate MTA configuration parameter. The CMS name is listed as **CMS1**.

In PacketCable, a single Server Kerberos principal name MAY be shared between multiple CMSs that MAY be located on different hosts. In that case, several CMS entries in an MTA configuration file will all have different DNS names or IP addresses but the same Kerberos principal name.

For full syntax of Kerberos principal names, refer to [18].

6.4.6.3.1 Service Key Versioning

The CMS service key that is shared between a TGS and CMS, to encrypt/decrypt CMS tickets, is a versioned key (refer to [17]). This key may be changed either due to a routine key refresh, or because it was compromised. When the CMS service key is changed, the CMS MUST retain the older key for a period of time that is at least as long as the ticket lifetime used when issuing CMS tickets (i.e. up to 7 days).

In the case of a routine service key change, the CMS MUST accept any ticket that is encrypted with an older key that it has retained and is still valid (not comprised). This key versioning on the CMS will prevent against many MTAs from suddenly flooding a TGS with PKINIT Requests for new tickets.

If a CMS service key is changed because it has been compromised, the CMS 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 CMS MUST respond as specified in [17] with a **KRB_AP_ERR_BADKEYVER** error. The CMS 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 6.5.5.1). 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 CMS ticket which is no longer valid and fetch a new one from its TGS.

The Kerberos Set/Change Password protocol (see [45]) SHOULD be used for setting and changing the CMS service keys. This protocol will provide for greater vendor inter-operability and facilitate key and password administration between the TGS and its principals. Routine CMS service key changes SHOULD be scheduled to occur autonomously using this protocol. Refer to Appendix E for recommendations on key refresh schedules.

6.4.6.4 Provisioning Server

The Provisioning Server possesses multiple principal names, where each one corresponds to a separate SNMPv3 user. A principal name for an SNMPv3 user MUST be of type NT-SRV-HST (3) with exactly two components, where the 1st component MUST be usmUserName (SNMPv3 user name) and the 2nd component MUST be the FQDN of the Provisioning Server:

<SNMPv3 user name>/<Prov Server FQDN>@<realm>

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

The $PKINIT_{GP}$ is not specified for the key management between the MTA and the Provisioning Server. When the MTA implementation requests a TGT in an AS Request and when the Provisioning Server realm is the same as the realm for one of the MTA signaling endpoints, the $PKINIT_{GP}$ value of that signaling endpoint 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 also 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.

6.4.7 MTA Principal Names

An MTA principal name MUST be of type NT-SRV-HST with exactly two components, where the 1st component MUST be the string "mta" (not including the quotes) and the 2nd component MUST be the FQDN of the MTA:

mta /<MTA FQDN>@<realm>

where <realm> is the Kerberos realm of the MTA and <MTA FQDN> is the MTA FQDN.

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

6.4.7.1 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 KDC MUST retrieve the mapping of the MTA MAC addresses to MTA FQDNs from the Provisioning Server. The actual interface between them is currently out of scope. The KDC MAY keep a local cache of the MTA MAC address -> MTA FQDN mappings.

6.4.8 Storage of MTA Public Keys

During initial MTA provisioning a SHA-1 hash of the MTA public key SHOULD be saved to the OSS database along with the time when the public key was saved. The mechanism used to store the public key hash is not specified in this version of PacketCable.

Once the public key hash is saved in the OSS database, the KDC SHOULD have a way to retrieve it and use it to verify the public key in the MTA Device Certificate in an AS Request (with PKINIT). This mechanism is particularly useful in the case where an MTA Manufacturer Certificate is revoked. In that case, the MTA Device Certificates issued under the revoked manufacturer certificate MAY still be accepted by the KDC – as long as the corresponding MTA public key hashes were saved in the OSS database some configurable period of time before the revocation.

In a normal case, where the MTA Manufacturer Certificate is still valid, the KDC SHOULD still retrieve the public key hash of the MTA and use it to validate the MTA Device Certificate. If the public key hash is not available, the KDC MUST follow the verification procedure specified by PKINIT.

6.4.9 Server Key Management Time Out Procedure

The MTA MUST perform a configurable backoff and resend procedure for any KDC or application server requests that have not been acknowledged by the server. The handling of this procedure and any subsequent fault MUST be consistent with the procedure described in Pkt-SP-EC-MGCP-D02-990207, Section 4.5.2.

If the MTA has reached the maximum number of retries with a particular KDC or Application Server IP address failing to get a reply, it MUST follow the failover procedure in section 6.4.9 of the NCS specification [3]. This procedure allows the MTA to retry with the alternative server IP addresses whenever a single server FQDN is mapped to multiple IP addresses.

6.4.10 Names of Other Kerberized Services

All Kerberized services within PacketCable MUST be assigned a service principal name of type KRB_NT_SRV_HST (Value=2), which has the following form according to the Kerberos specification:

<service name>/<FQDN>

This means that the last component of the service principal name is always the FQDN of the corresponding host.

6.4.11 Kerberos Cross-Realm Operation

It is possible that Security Parameters will have to be established between two entities in two different administrative domains. Each PacketCable administrative domain corresponds to a Kerberos realm.

Kerberos key management requires that one of these entities (acting as a client) obtain a service ticket from the remote KDC (in the server's realm). In the case that the client entity does not use PKINIT, it cannot authenticate directly to the remote KDC.

The client MUST first obtain a cross-realm TGT for the remote KDC from its local KDC. Then, the client MUST authenticate to the remote KDC with this cross-realm TGT in order to obtain a service ticket for the remote server. And, in order for the local KDC to issue a cross-realm TGT there needs to be some trust established between the two realms.

There are two ways within Kerberos to establish trust between two realms. The 1st way is to establish a pre-shared key between the two realms, called a cross-realm key. The 2nd way is the use of public key authentication and automatic establishment of the cross-realm key via the PKCROSS protocol, see Appendix C. PKCROSS utilizes PKINIT for establishing the inter-realm key and associated inter-realm policy to be applied in issuing cross realm service tickets between realms and domains in support of Intradomain and Interdomain CMS-to-CMS signaling (CMSS). A PacketCable KDC MUST support PKCROSS and MAY also support pre-shared cross-realm keys.

Section 6.4.1 outlines the various phases of Kerberos-based key management. Whenever cross-realm authentication is involved, that introduces a new key management phase. This phase would run after phase 1 and before phase 2 – we will call it phase 1.5. In phase 1.5, a client in realm A requests a cross-realm TGT for realm B from its local KDC. If a cross-realm key doesn't already exist between realms A and B, the local KDC MUST automatically run PKCROSS to establish the cross-realm key. The local KDC (Realm A) MUST then return the requested cross-realm TGT to the client.

Phase 2 will proceed as before, except that the client in realm A authenticates itself to the remote KDC in realm B with the cross-realm TGT obtained in phase 1.5, rather than with the TGT obtained in phase 1. Phase 3 in this case is unchanged.

Kerberos cross-realm authentication is illustrated in the diagram below:

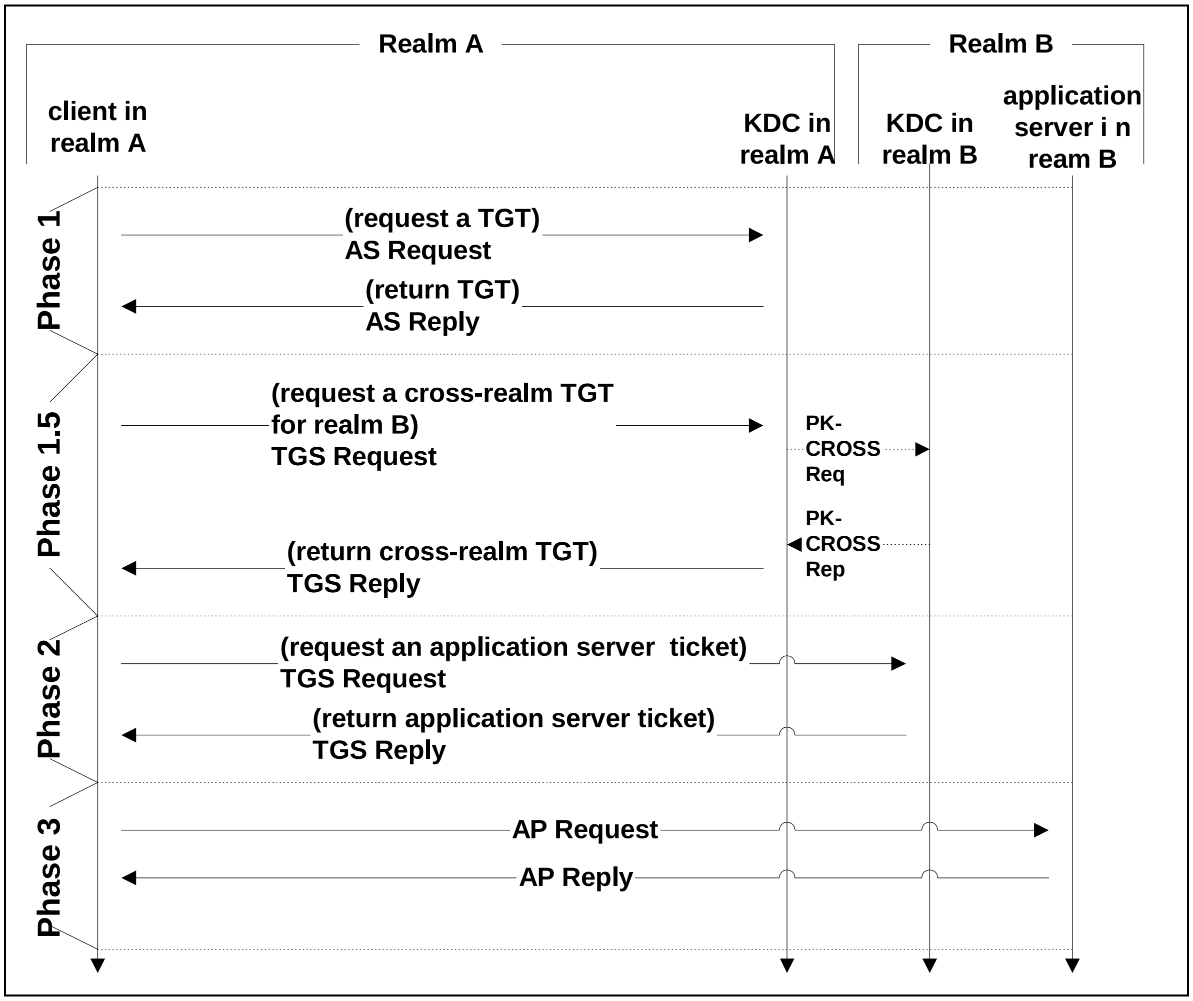
Realm A
Realm B
client in realm A
KDC in realm A
KDC in realm B
application server i n ream B
Phase 1
(request a TGT) AS Request
(return TGT) AS Reply
Phase 1.5
(request a cross-realm TGT for realm B) TGS Request
PK-CROSS Req
PK-CROSS Rep
(return cross-realm TGT) TGS Reply
Phase 2
(request an application server ticket) TGS Request
(return application server ticket) TGS Reply
Phase 3
AP Request
AP Reply

Figure 8. Kerberos Cross-Realm Operation

6.4.11.1 PacketCable Profile for Cross-Realm Operation

There is a new exchange required by the cross-realm operation in phase 1.5, where the client obtains a cross-realm TGT. A client MUST request a cross-realm TGT with a TGS Request. The PacketCable profile for a TGS Request/TGS Reply exchange is specified in section 6.4.5. The only difference is that the cross-realm TGT is not encrypted with the normal TGS key of the local KDC – instead it is encrypted with a cross-realm key (see [18]).

In the case where a cross-realm key doesn't already exist, the TGS Request for a cross-realm TGT MUST trigger a PKCROSS exchange between the two KDCs, resulting in the automatic establishment of the cross-realm key. This is specified in the section 6.4.11.3.

6.4.11.2 Referrals

In the above diagram, in phase 1.5 the client might not know the realm of the remote server and thus will be unable to generate a TGS request for a cross-realm TGT.

For PacketCable, when the client does not know the realm of the server, it MUST assume that it is in the local realm and send a TGS request for the service ticket to the local KDC. In this case, the local KDC MUST attempt to determine the realm of the server with a query for a DNS SRV record. Below is an example of the DNS SRV records that map host names to a Kerberos realm:

_kerberos.asdf.com. IN TXT "ASDF.COM"

_kerberos.CMS1.asdf.com. IN TXT "FOO.ASDF.COM"

Let us suppose that in this case, the KDC gets a TGS Request for the service on the host CMS2.asdf.com. It would first query:

_kerberos.CMS2.asdf.com. IN TXT

Finding no match, it would then query:

_kerberos.asdf.com. IN TXT

And find an answer of ASDF.COM. This would be the realm that CMS2.asdf.com resides in.

If another TGS Request asks for the Kerberized service on the host CMS1.asdf.com, the KDC would query:

_kerberos.CMS1.asdf.com IN TXT

And find an answer of FOO.ASDF.COM.

Once the KDC had obtained the remote realm name, it MUST obtain a cross-realm key with that realm, which may require a PKCROSS exchange described in the following section. Once the KDC has established cross-realm keys with the remote KDC, it MUST issue the cross-realm ticket for the remote realm and MUST return it in the TGS Reply. (This case, where the local KDC returns a cross-realm TGT to the client instead of the requested service ticket, is called a referral in the Kerberos specification.)

6.4.11.3 PKCROSS Exchange

Kerberos cross realm authentication requires that administrators maintain separate keys for every realm for which a direct trust relationship is possible. Indirect, transitive trust is also possible, but it relies on trust of intermediate realms, and it is unnecessarily complex due to location of intermediate realms and establishment of transitive trust policies. For more information on Kerberos transitive trust issues, see [17]. Direct trust relationships require n(n-1) keys to be established and administered, which rapidly becomes an unwieldy administrative burden for maintaining keys and policies.

PKCROSS leverages a public key infrastructure (PKI) to establish trust between Kerberos realms, while it mitigates administrative issues of PKI by limiting the number of PKI endpoints to just Kerberos realms. In this way, Kerberos may be utilized for key management of large numbers of centrally administered principals, while PKI may be utilized for inter-realm key management of a small number of

Kerberos realms. Thus, PKCROSS enables the dynamic establishment of cross realm Kerberos keys. This exchange uses the PKINIT protocol and enables a remote Kerberos realm to issue a cross realm key and policy information to another realm. This key and policy information is then returned to the remote realm in the form of a special cross realm ticket.

The figure above, depicts the flows for cross realm authentication. The following description explains more of the PKCROSS exchange:

Table 6. PKCROSS Exchange

KDC_L	local KDC
KDC_R	remote KDC
$XTKT_{(L,R)}$	PKCROSS ticket that the remote KDC issues to the local KDC – contains the dynamic Cross-Realm key
$TGT_{(C,R)}$	cross-realm TGT that the local KDC issues to the client for presentation to the remote KDC

Phase 1.5:

KDC_L issues a PKINIT request to KDC_R with the PKCROSS flag (bit 9) set in the AS-REQ kdc-options field.

KDC_R replies with $XTKT_{(L,R)}$ that is encrypted under KDC_R's PKCROSS key. Note that, within the PKCROSS protocol, the PKCROSS key, as defined in the PKCROSS specification, is used in place of the TGS key. Also, KDC_R MAY place policy information in ticket extensions (for example, this policy information may reflect service level agreements)

KDC_L applies the policies dictated by KDC_R in $XTKT_{(L,R)}$ and it issues $TGT_{(C,R)}$ to the client. This $TGT_{(C,R)}$ is encrypted under the key that resides in $XTKT_{(L,R)}$. $XTKT_{(L,R)}$ is added to $TGT_{(C,R)}$ as a ticket extension. (TicketExtensions is an optional field in a Kerberos Ticket).

Phase 2:

When the client presents $TGT_{(C,R)}$ to KDC_R, KDC_R extracts $XTKT_{(L,R)}$ and is then able to decrypt $TGT_{(C,R)}$ and verify policy on the ticket, then it issues a service ticket to the client.

6.4.11.4 Determining the Location of a Remote KDC

The FQDN of a remote KDC is determined with a DNS SRV record lookup. This mechanism is identical to the mechanism used by a client to locate a KDC in the local realm.

6.5 Kerberized Key Management

6.5.1 Definitions

Security Parameters: A security relationship established according to the protocol for which Kerberos is being used for key management (e.g. IPSec, SNMPv3).

6.5.2 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. The subkey from the AP Reply is 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).

6.5.3 Kerberized Key Management Messages

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

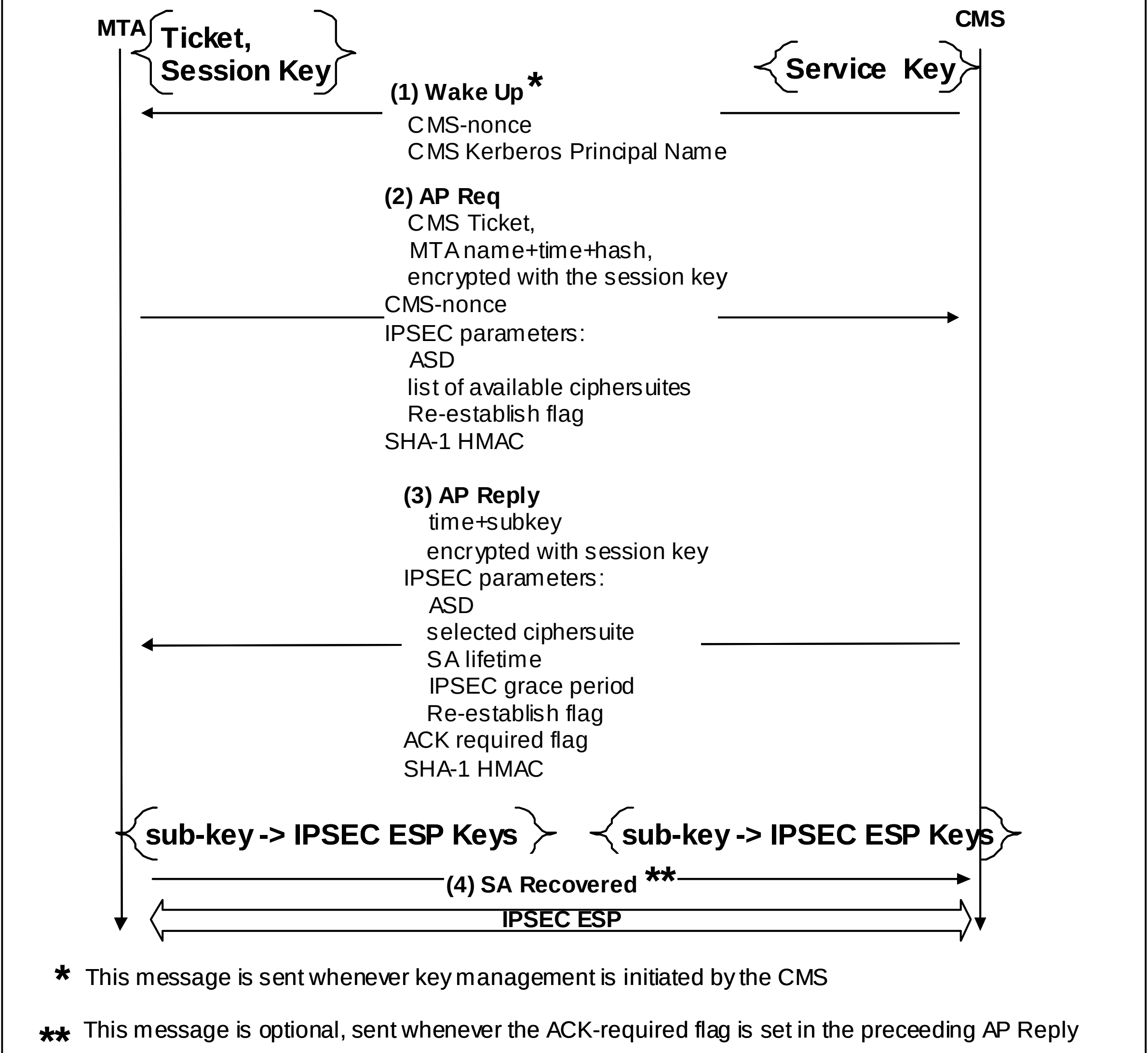
MTA
Ticket, Session Key
CMS
Service Key
(1) Wake Up *
CMS-nonce
CMS Kerberos Principal Name
(2) AP Req
CMS Ticket,
MTA name+time+hash,
encrypted with the session key
CMS-nonce
IPSEC parameters:
ASD
list of available ciphersuites
Re-establish flag
SHA-1 HMAC
(3) AP Reply
time+subkey
encrypted with session key
IPSEC parameters:
ASD
selected ciphersuite
SA lifetime
IPSEC grace period
Re-establish flag
ACK required flag
SHA-1 HMAC
sub-key -> IPSEC ESP Keys
sub-key -> IPSEC ESP Keys
(4) SA Recovered **
IPSEC ESP
* This message is sent whenever key management is initiated by the CMS
** This message is optional, sent whenever the ACK-required flag is set in the preceeding AP Reply

Figure 9. 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 name, 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 name**: a printable, null-terminated ASCII string, representing a fully qualified Kerberos Principal Name of the Application Server, as specified in [18].

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 6.4.9. When the "Timeout Procedure" has completed without success, the Application Server MUST discard this server-nonce, 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, it MUST respond with this AP Request.

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 6.5.6.4. 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 6.5.6.5.

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 [18].
- **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 6.5.4, 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 Security Parameter before the old one expires. 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 name 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 6.4.9 .

If the AP Request was generated in response to a message sent by the Application Server (Wake Up or Rekey), then the client MUST keep the nonce and server Kerberos principal name for pktcClntSolicitedKeyMgmtTimeout (for the MTA it is the pktcMtaDevSolicitedKeyMgmtTimeout MIB variable for IPSec key management). After the timeout has been exceeded or when the "Timeout Procedure" has completed without success, the client MUST discard this (nonce, server Kerberos principal name) 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 6.4.9.

In the case that the Server-nonce is 0 (not filled in), the Application Server MUST verify that this AP Request is not a replay using the procedure specified in the Kerberos standard [17]:

- If the timestamp in the AP Request differs from the current Application Server time by more than **pktcSrvrToMtaMaxClockSkew** then Application Server MUST reply with an error message specified in section 6.5.5.2.
- 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 is returned. The Application Server MUST remember any authenticator

presented within pktcSrvrToMtaMaxClockSkew, so that a replay attempt is guaranteed to fail.

- If the Application Server loses track of any authenticator presented within pktcSrvrToMtaMaxClockSkew, it MUST reject all requests until the clock skew 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 [17]. 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 6.5.3.
- **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 [18].
- **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.
- **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 as specified in section 6.5.6.2.1. When the value is FALSE (0), the old set of security parameters MUST be allowed to expire as specified in section 6.5.6.3.
- **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 this HMAC. If this integrity check fails, the Client MUST immediately discard the AP Reply and 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 **pktcSrvrKeyMgmtTimeout3** in order to validate a Security Parameter Recovered response from the Client. After pktcSrvrKeyMgmtTimeout3 the Application Server MUST discard this value, after which it will no longer accept a matching Security Parameter Recovered.

(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 6.5.3.
- **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.

6.5.4 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 an 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:

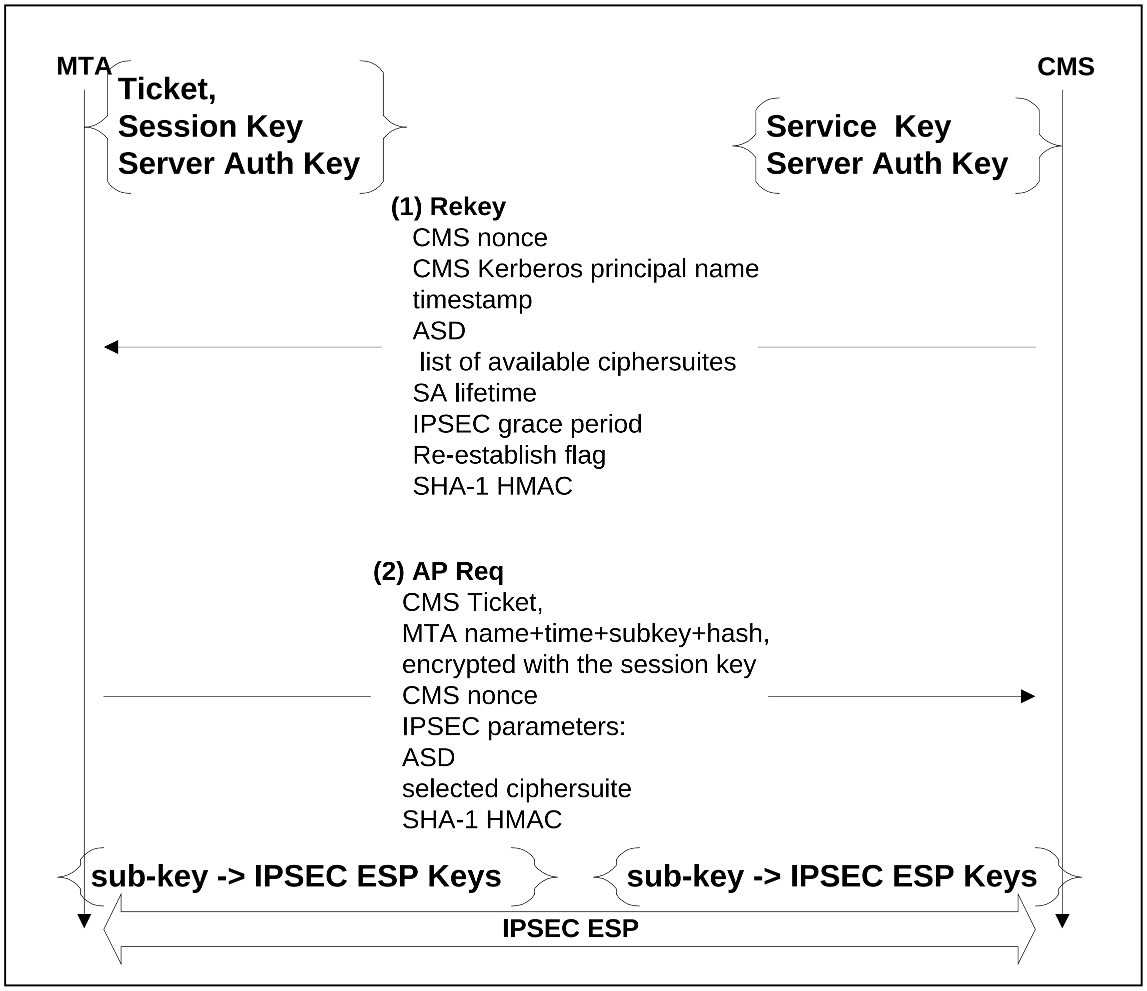

Figure 10. Rekey Message to Establish a Security Parameter

The messages listed in this diagram are defined as follows:

(1) Rekey – sent by the Application Server to establish a new Security Parameter. 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 6.5.3.
- **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 Name** – a printable, null-terminated ASCII string, representing a fully qualified Kerberos Principal Name of the Application Server, as specified in [18]. 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 or SNMPv3).
- **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 as specified in section 6.5.6.3. When the value is FALSE (0), the old set of security parameters MUST be allowed to expire as specified in section 6.5.6.2.2.
- **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.)[3]

[3] 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.

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 6.4.9. When the "Timeout Procedure" has completed without success, the Application Server MUST discard the server-nonce, 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 [17]:

- If the timestamp in the Rekey message differs from the current Client time by more than **pktcSrvrToMtaMaxClockSkew** then the Client MUST drop the message.
- 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 **6.4.4**. 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.

Both of these error cases could be completely avoided with a 3-way handshake (a Client acknowledging an AP Reply with an SA Recovered message), which is not used in this case for performance reasons – to avoid an extra upstream message.

6.5.5 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 are not supported***.

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**: random value generated by the Client

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.[4] The actual subkey length is dependent on the target protocol.

All other optional fields within the encrypted authenticator are not supported within PacketCable. The authenticator itself MUST be encrypted using 3-DES CBC with the Kerberos **etype** value**: des3-cbc**. PacketCable does not use combined etypes that specify an encryption algorithm and a checksum, since a keyed HMAC (outside of the KRB_AP_REP) is used instead.

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**.

6.5.5.1 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.

[4] 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.

- **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.

- **Domain of Interpretation (DOI)** – 1 byte value. Specifies the target protocol for which security parameters are established. See Table in section 6.5.3.

- **KRB-ERROR** – Kerberos error message as specified in [18]. 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. 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 des3-cbc-md5, as it is specified in [17]. This keyed checksum is calculated by:

  (1) take an MD5 hash of the **KRB-ERROR** message

  (2) prepending the hash with an 8-byte random byte sequence, called a **confounder**

  (3) take the 3DES session key from the ticket and XOR each byte with F0

  (4) use 3DES in CBC mode to encrypt the result of step 2, using the key in step 3 and with IV(initialization vector)=0

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 [17]):

```
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 it's 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.

6.5.5.2 Clock Skew Error

When the Application Server clock and the client clock are off by more than the limit for a clock skew (usually 5 minutes), an error code KRB_AP_ERR_SKEW MUST be returned. 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 use it to adjust its clock in subsequent PKINIT and AP Request messages.

In the case that an AP Request failed due to a clock skew error, an Client MUST immediately retry after adjusting its clock.

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 an Client into accepting an expired KDC certificate (later, during the next PKINIT exchange).

6.5.6 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 Association are always uni-directional and they are always established in pairs within PacketCable.

The DOI value for IPSec MUST be set to 1.

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

The Application-specific data field in the AP Reply and Rekey key management messages MUST be the ASD for the server's inbound Security Associations. It is a 4-byte integer value, MSB first.

The subkey for IPSec MUST be a 46-byte value

6.5.6.1 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:

a) Message authentication key for Client->Application Server messages

b) Encryption key for Client->Application Server messages

c) Message authentication key for Application Server->Client messages

d) 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.

The derivation of the required keying material MUST be based on running a one-way pseudo-random function **F(S, "IPSec Security Associations")** 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 9.5.

6.5.6.2 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.

6.5.6.2.1 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. After the Client receives an AP Reply, it MUST perform the following steps:

(1) Create new IPSec SAs, based on the negotiated ciphersuite, ASDs and on the established Kerberos subkey, from which the IPSec keys are derived as specified in section 6.4.4. 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 an 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.

6.5.6.2.2 Periodic Re-establishment of IPSec SAs at the Application Server

When an AP Request message is received and right before an AP Reply is returned, the Application Server MUST perform the following steps, in the specified order.

(1) create new IPSec SAs, based on the negotiated ciphersuite, ASDs 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 TEXP, 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.

6.5.6.3 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 IPSec layer MUST signal the key management layer to establish a new IPSec SA. It is established using the same procedures as the ones specified in section 6.5.6.5.

6.5.6.4 Initial Establishment of IPSec SAs

When an 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.

6.5.6.5 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.

6.5.6.5.1 Client Loses an Outgoing IPSec SA

When an Client attempts to send a signaling message to the Application Server without a valid IPSec SA. At that time, the IPSec layer in the Client realizes the SA is missing and returns an error back to the signaling application.[5] 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 6.4.3.

(2) Client sends a new AP Request to the Application Server and gets back an AP Reply, as specified in section 6.5.4.

After the receipt of the AP Reply the Client MUST be prepared to use both of the newly created IPSec SAs.

The Application Server MAY set an ACK-required flag in the AP Reply. In that case, right after sending out an AP Reply, the Application Server MUST be prepared to receive messages on the incoming SA but cannot yet start using an outgoing SA for sending messages to the Client. In this case, the IPSec SA setup continues with the following steps 3 and 4.

The Application Server also MAY NOT set the ACK-required flag in the AP Reply. In that case, right after sending out an AP Reply, the Application Server MUST be prepared to both send and receive messages on the newly created SAs. In this case, steps 3 and 4 below are skipped.

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.

(3) Immediately after the Client establishes the new IPSec SAs, it sends a **SA Recovered** message to the Application Server.

(4) Upon receipt of this message, the Application Server will immediately activate the new outgoing SA for sending signaling messages to the Client.

[5] In this case, there are no actual messages exchanged between the MTA and the CMS.

The key management application running on the Client MUST send an explicit signal when it completes the re-establishment of the IPSec SAs. The signaling application at the Client MUST retry sending an AP Request after some period of time when it has not received an explicit signal from the key management application running on the same Client, when it completes the establishment of IPSec SAs.

6.5.6.5.2 Client Loses an Incoming IPSec SA

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

6.5.6.5.3 Application Server Loses an Outgoing IPSec SA

When a 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.[6] 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. For each AP Request, the Client generates a nonce and puts it into the **seq-number** field. As specified in section 6.5, the Client will save this nonce for a short period of time 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 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. 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 6.5.6.3.

[6] In this case, there are no actual messages exchanged between the MTA and the CMS.

In the event that 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.

(4) Immediately after the Client establishes the new IPSec SAs, it MUST send a **SA Recovered** message to the Application Server.

(5) 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 when it completes the re-establishment of the IPSec SAs. The signaling application at the Application Server MUST retry a WAKE UP send after some period of time when it has not received an IPSec SA established signal.

6.5.6.5.4 Application Server Loses an Incoming IPSec SA

When the Application Server receives an IP packet from an 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.

6.5.7 Kerberized SNMPv3

This section specifies the Kerberized key management profile specific to SNMPv3, see [41]. In the case of SNMPv3, the security parameters are associated with the usmUserName (SNMPv3 user name) on the server (SNMP Manager) side and with the usmUserEngineID (SNMPv3 engine ID) on the client (agent) side.

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 and a single privacy (encryption) key. Within PacketCable, SNMPv3 authentication MUST always turned on. In addition, SNMPv3 privacy MAY also be used (it can be turned off by selecting a NULL encryption transform).

The DOI value for SNMPv3 MUST be set to 2.

The Application-specific data field in the AP Request and AP Reply key management messages MUST be set to the SNMPv3 Engine ID corresponding to the SNMP agent. The SNMP Manager MUST verify the SNMPv3 Engine ID contained in the AP Request based on the contents of the client principal name contained in the ticket. For PacketCable MTAs, the manager MUST verify that the MTA FQDN specified in the principal name corresponds to the SNMPv3 Engine ID. The SNMPv3 Engine ID 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.

6.5.7.1 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 is 46 bytes.

The derived SNMPv3 keys are 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 consists of running 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 **9.5**.

6.5.7.2 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.

6.5.7.3 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.

6.5.7.4 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 6.5.7.1.

6.5.7.5 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.

6.5.7.5.1 SNMP Agent Wishes to Send with Missing SNMPv3 Keys.

An SNMP agent is capable of initiating protocol exchanges with the manager, e.g. with the SNMP Trap and SNMP Inform messages. If the SNMP agent determines that it is missing 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 **6.4.3**.

(2) The agent sends a new AP Request to the manager and gets back an AP Reply, as specified in section 6.4.4

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.

6.5.7.5.2 SNMP Agent Receives with Missing SNMPv3 Keys

The SNMP agent receives a request from a manager and is unable to find SNMPv3 keys for the specified user. The agent MUST ignore this error and the message MUST be dropped. In this case, any attempt at recovery (e.g., establishing new SNMPv3 keys) is prone to denial of service attacks.

6.5.7.5.3 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 6.5.3. For each AP Request, the agent generates a nonce and puts it into the **seq-number** field. As specified in section 6.5.6.5.3, the agent will save this nonce for a short period of time 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 6.5.3. 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 for the same SNMPv3 user name before the first key management session has been completed, the SNMP agent MUST ignore the second Wake Up message.

6.5.7.6 IPSec-Specific Errors Returned in KRB-ERROR

Inside AppSpecificTypedData the oid field MUST be set to <TBD>. 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.

6.5.7.7 SNMPv3-Specific Errors Returned in KRB-ERROR

Inside AppSpecificTypedData the oid field MUST be set to <TBD>. 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 6.6. 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 7. RTP Packet Transform Identifiers

Transform ID	Value	Key Size (in bits)	MUST Support	Description
reserved	0x50	-	-	
RTP_RC4	0x51	128	yes	RC4 stream cipher
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_RC4 Transform ID MUST be supported.

Table 8. 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 6.6)	yes	2-byte MMH MAC
reserved	0x63	-	-	
RTP_MMH_4	0x64	variable (see section 6.6)	yes	4-byte MMH MAC
reserved	0x65	-	-	

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

6.7 BPI+

All Clients MUST use DOCSIS 1.1 compliant cable modems and MUST implement BPI+ [14]. 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.

6.8 Radius

Radius protocol requires an authenticator field for all messages, which provides message integrity. No other security services or key management are defined within the Radius standard [21].

A 16-byte Authenticator field is calculated as follows:

- **Request Authenticator**: MD5 hash calculated over a stream of octets consisting of the Request Code + Identifier + Length + 16 zero octets + request attributes + shared secret (where + indicates concatenation).
- **Response Authenticator:** MD5 hash calculated over a stream of octets consisting of the Response Code + Identifier + Length + Request Authenticator field from the Accounting-Request packet being replied to + the response attributes if any + shared secret

The shared secrets for the Response and Request Authenticator fields do not have to be the same.

PacketCable interfaces that utilize Radius require that the authentication algorithm (ciphersuite) be specified (see section 6.1.2.2). Currently, only the standard Radius authentication mechanism (as described above) is supported and the ID for this authentication algorithm is **100** (decimal).

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 describes 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 per-protocol security analysis is organized by functional categories (see section 5.2.1.5). 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 9. 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*	*IPSec ESP in transport mode with both encryption and message integrity enabled. The UDP check sum may need to be turned off (set to*

	RTP (MTA – MTA, MTA – PSTN GW)	RTCP (MTA – MTA, MTA – MG, MG – MG)
	RC4-128 encryption algorithm *Optional 2-byte or 4-byte MAC based on MMH algorithm* *PacketCable requires support for ciphersuite negotiation.*	*0), depending on how NAT is implemented.* *Keys derived from bearer channel RTP SA*

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. Since a CMS can perform multiple functions, the security profile matrices indicate which of the CMS functional components is participating in the identified protocol interface.

7.1 Device and Service Provisioning

Note: This section subject to change significantly with ECR, sec-r-00138, currently in process.

Device provisioning is the process by which an MTA is configured to support voice communications service. The MTA provisioning process is specified in [6].

The following figure illustrates the flows involved with the provisioning processes. The provisioning specification lays these flows out in detail. The flows involving security mechanisms are described in this section of the document.

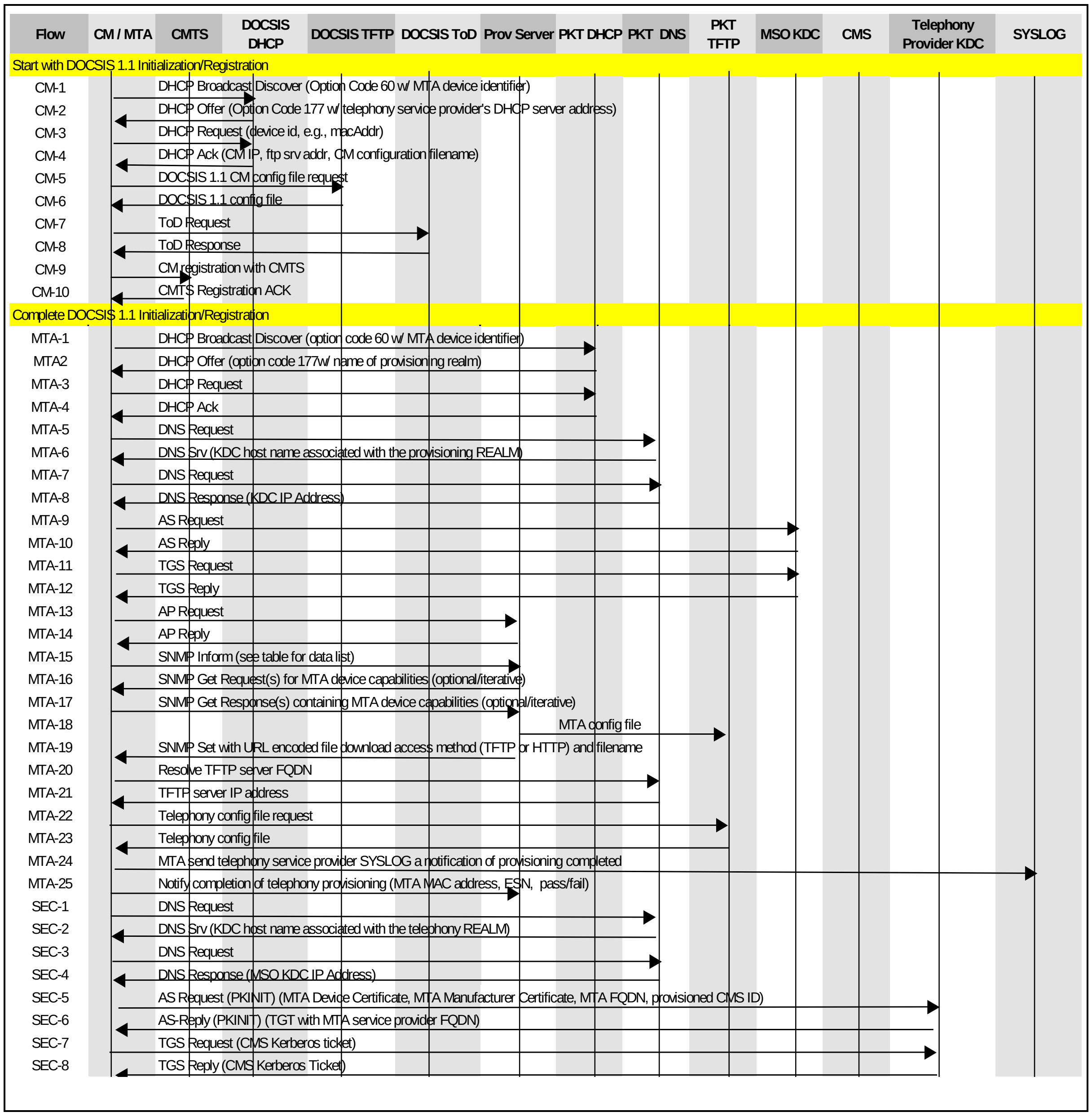
Flow
CM / MTA
CMTS
DOCSIS DHCP
DOCSIS TFTP
DOCSIS ToD
Prov Server
PKT DHCP
PKT DNS
PKT TFTP
MSO KDC
CMS
Telephony Provider KDC
SYSLOG
Start with DOCSIS 1.1 Initialization/Registration
CM-1 DHCP Broadcast Discover (Option Code 60 w/ MTA device identifier)
CM-2 DHCP Offer (Option Code 177 w/ telephony service provider's DHCP server address)
CM-3 DHCP Request (device id, e.g., macAddr)
CM-4 DHCP Ack (CM IP, ftp srv addr, CM configuration filename)
CM-5 DOCSIS 1.1 CM config file request
CM-6 DOCSIS 1.1 config file
CM-7 ToD Request
CM-8 ToD Response
CM-9 CM registration with CMTS
CM-10 CMTS Registration ACK
Complete DOCSIS 1.1 Initialization/Registration
MTA-1 DHCP Broadcast Discover (option code 60 w/ MTA device identifier)
MTA2 DHCP Offer (option code 177w/ name of provisioning realm)
MTA-3 DHCP Request
MTA-4 DHCP Ack
MTA-5 DNS Request
MTA-6 DNS Srv (KDC host name associated with the provisioning REALM)
MTA-7 DNS Request
MTA-8 DNS Response (KDC IP Address)
MTA-9 AS Request
MTA-10 AS Reply
MTA-11 TGS Request
MTA-12 TGS Reply
MTA-13 AP Request
MTA-14 AP Reply
MTA-15 SNMP Inform (see table for data list)
MTA-16 SNMP Get Request(s) for MTA device capabilities (optional/iterative)
MTA-17 SNMP Get Response(s) containing MTA device capabilities (optional/iterative)
MTA-18 MTA config file
MTA-19 SNMP Set with URL encoded file download access method (TFTP or HTTP) and filename
MTA-20 Resolve TFTP server FQDN
MTA-21 TFTP server IP address
MTA-22 Telephony config file request
MTA-23 Telephony config file
MTA-24 MTA send telephony service provider SYSLOG a notification of provisioning completed
MTA-25 Notify completion of telephony provisioning (MTA MAC address, ESN, pass/fail)
SEC-1 DNS Request
SEC-2 DNS Srv (KDC host name associated with the telephony REALM)
SEC-3 DNS Request
SEC-4 DNS Response (MSO KDC IP Address)
SEC-5 AS Request (PKINIT) (MTA Device Certificate, MTA Manufacturer Certificate, MTA FQDN, provisioned CMS ID)
SEC-6 AS-Reply (PKINIT) (TGT with MTA service provider FQDN)
SEC-7 TGS Request (CMS Kerberos ticket)
SEC-8 TGS Reply (CMS Kerberos Ticket)

Figure 11. PacketCable Provisioning Flows

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.

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, where an MTA is 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

All SNMP traffic between the MTA and the SNMP Manager in both directions is protected with SNMPv3 security [41]. PacketCable requires that SNMPv3 message authentication is always turned on with privacy being optional. 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 reference [41] for the list of SNMPv3 cryptographic algorithms supported by PacketCable.

7.1.1.1.3 MTA-Provisioning Server, via TFTP Server

Authentication: the identity of the OSS that generated the MTA configuration file is authenticated with comparing the hash of the configuration file that was generated by the Provisioning Server and transported to the MTA securely via SNMPv3 against the hash of the configuration file downloaded from the TFTP server. This is required to prevent denial of service attacks, where an MTA is improperly configured.

The identity of the MTA requesting the file is not authenticated. Authentication of the MTA is not required; if there is a need to keep the file contents private, it may be encrypted with the Provisioning Server-MTA shared key and no one else will be able to use it.

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

Confidentiality: is 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: is not required.

7.1.1.2 Cryptographic Mechanisms

7.1.1.2.1 Call Flow MTA-15: 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 [41]. PacketCable requires that SNMPv3 message authentication is always turned on with privacy being optional. 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 6.5.7 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

In this flow, the Provisioning Server builds a 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 do the following steps:

1. If 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 (1). 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 (2) 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 configuration file format is described in section 7.1.1.2.2 above.

7.1.1.2.5 Security Flows

The following security flows MUST be performed immediately following the provisioning process. These flows MUST be performed for every entry in the table PktcMtaDefCmsTable.

Table 10. 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 Request The MTA requests the Telephony KDC host name for the kerberos realm.	SEC-1
SEC-2	DNS Srv Returns the Telephony KDC host name associated with the provisioning REALM.	SEC-1
SEC-3	DNS The MTA now requests the IP Address of the Telephony KDC.	SEC-1
SEC-4	DNS The DNS Server returns the IP Address of the Telephony KDC.	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 containing the MTA Device Certificate and the MTA FQDN.	Report alarm. Abort establishment of signaling security.
SEC-6	AS Reply The KDC sends the MTA a PKINIT REPLY message containing the requested Kerberos Ticket.	SEC-5 or report alarm and abort signaling security depending upon error condition.

Sec Flow	Flow Description	If Step Fails, Proceed Here
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.	Report alarm. Abort establishment of signaling security.
SEC-8	TGS Reply Response to TGS Request containing the requested CMS Kerberos Ticket.	SEC-7, SEC-5 or report alarm and abort signaling security depending upon error condition.
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.	Report alarm. Abort establishment of signaling security. Or go to SEC-9 ,SEC-7, or SEC-5 depending on the error condition.
SEC-11	The MTA responds with an SA Recovered message that lets the CMS know, the MTA is now ready to receive on its incoming IPSec Security Association. This message is sent when requested by the flag in the AP Reply. This flag should not be used in the initial provisioning flows.	Report alarm. Abort establishment of signaling security.

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). 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 ticket. It is also possible, that the MTA is configured to request Kerberos Tickets from different Telephony KDC servers, depending on the CMS group.

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 sets of SAs will be established (one set for each CMS).

Since a Kerberos Ticket is issued for a group of CMSs, it is possible that a single Kerberos Ticket is used to establish more than one set of IPSec SAs.

In PacketCable, 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 initially establish SAs 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 6.5.7. The MTA and the Provisioning Server MUST support this key management protocol. Additional SNMPv3 users MAY be created with the standard SNMPv3 cloning method [41] 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 6.4.6.1). When there is more than one KDC (DNS SRV record) found, DNS assigns a priority to each one. The MTA will choose a KDC based on the DNS priority 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

Table 11. 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 MTA's MAC address. 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 7.4 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

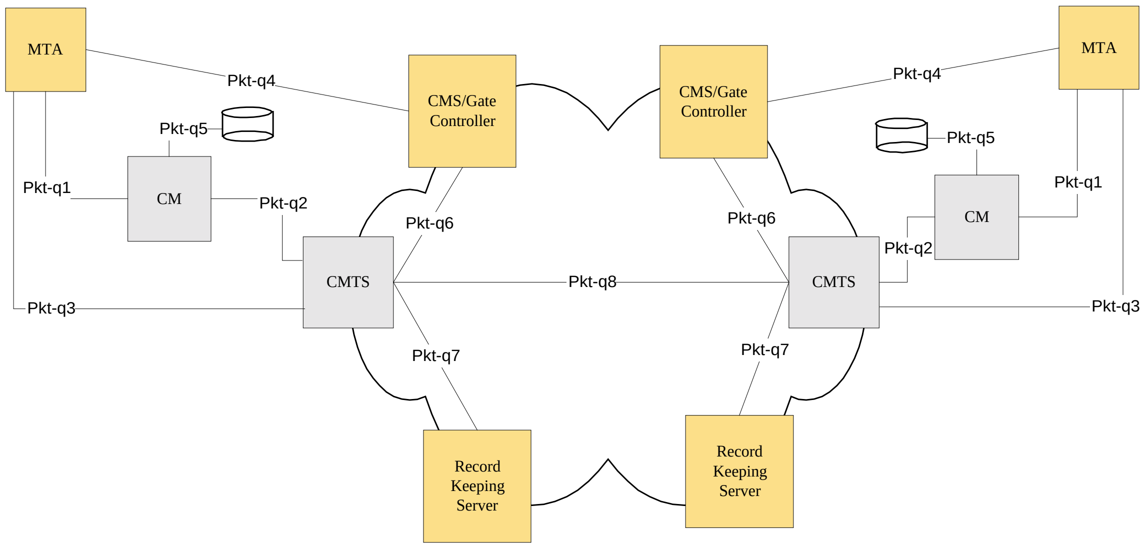

Figure 12: QoS Signaling Interfaces in PacketCable Network

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 [11].

7.2.1.2.2 CMTS-*CMS Gate Coordination Messages (over UDP)*

The CMTS-CMS Gate Coordination messages are required in order to prevent some theft of service scenarios, described in the DQoS specification [3].

Authentication: required to prevent theft of service and denial of service attacks. Direct authentication is not possible due to the large number of associations and performance requirements (for post-dial and post-pickup delays). Each end is indirectly authenticated, via a trusted third party – the CMS.

Message Integrity: required on this interface. Without it, the same service theft scenarios are still possible, along with the denial of service attacks.

7.2.1.2.3 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 [11].

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.

A Classifier is a set of matching criteria applied to each packet entering the cable network. A classifier 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 from flooding the network with unauthorized IP packets.

To address DOCSIS 1.1 CMTS accesses to the network, the CMTS SHOULD apply upstream classifiers to police upstream packets from its network; including the verification of the source IP address.

For more information regarding the use of packet classifiers, refer to the DOCSIS 1.1 RFI spec [11].

7.2.1.3.2 CMTS-CMS Gate Coordination Messages (over UDP)

These are DQoS handshake messages between the two sides of the communication, to ensure that the QoS resources had been reserved on both sides. DQoS handshake messages MUST be formatted as Radius messages, which MUST include an authenticator field. Security for this interface is provided solely by the Radius-specific authenticator, based on an MD5 hash, as defined by RFC 2139 [21]. This provides message integrity, but not privacy. The key for the Radius authenticator MUST be exactly 16 bytes long, and there MUST be a separate key used for each direction.

7.2.1.3.3 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 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 CMTS-CMS Gate Coordination Messages (over UDP)

The keys for this interface MUST be securely distributed by the local CMS over the existing IPSec links. The key MUST be included in the Gate Authorization message (COPS) sent from the local CMS to the CMTS. For the on-net-to-on-net communications, the keys for Gate Coordination messages on each side of the communication are different.

There is only one message that is involved in the key management for this interface: the Gate-Set Authorization message sent to the CMTS by the local CMS, which MUST include the following parameters:

- Transaction ID (associates request an response together)

Gate ID

Authentication algorithm (1 byte) – only Radius, MD5-based MAC is currently supported. See section 6.8.

Radius Key (16 bytes)

- IP address of the local CMS (listed as the IP address of the remote gate in the Gate-Set messages).

For the gate coordination exchange to proceed, the CMTS MUST accept this Radius Key and ciphersuite and MUST use it to authenticate Gate Coordination messages in both directions for the specified Gate ID.

To address communication keying replay protection concerns, the same Radius key is used to authenticate only a few messages. These messages consist of the Gate-Open/Gate-Open-Ack and the Gate-Close/Gate-Close-Ack exchanges. Each message type is clearly identified and appears exactly once for each communication.

7.2.1.4.2 Gate Controller – CMTS COPS Messages

Key management for this COPS interface MUST be implemented via IKE and use pre-shared keys. For more information on the PacketCable use of IKE, refer to section 6.2.2.

7.2.1.5 Summary Security Profile Matrix

Table 11. Security Profile Matrix – DQoS

	COPS (CMTS-CMS)	**Gate Coordination (CMTS – CMS)**
authentication	yes	yes (through a 3rd party)
access control	yes	no
integrity	yes	yes
confidentiality	yes	no
non-repudiation	no	no
Security Mechanisms	*IPSec with encryption and message integrity* IKE w/ pre-shared keys	*Radius authentication (MD5-based MAC)* CMS distributes key per communication

7.2.2 Interdomain QoS

Interdomain QoS consists of two different mechanisms, Differentiated Services (DiffServ) and an admission control protocol called RSVP.

7.2.2.1 Architecture Overview

The overall IQoS network architecture is depicted in the Figure below. The backbone consists of a general topology managed IP network that may comprise multiple administrative domains.

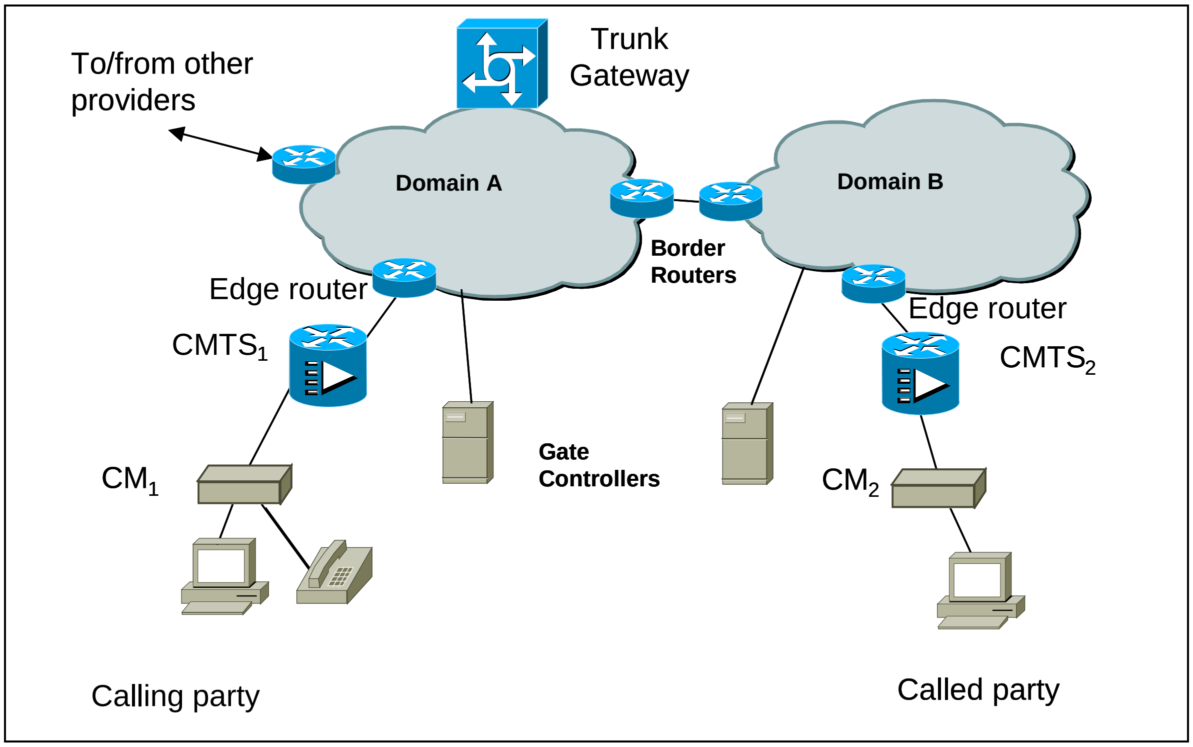

Figure 13. Interdomain QoS Architecture

In this architecture, we assume that DQoS signaling is used in the access network. At a minimum, the backbone is expected to be compliant with the DiffServ architecture. Border routers are those that sit at the boundaries between providers. They have specific roles in a DiffServ environment (such as aggregate policing and re-marking) that are discussed in more detail in the sections that follow.

7.2.2.2 Differentiated Services (DiffServ)

DiffServ allows IP traffic to be marked with different DiffServ Code Points (DSCP) to obtain different queuing treatment on routers. Different queuing treatments in each router are called per hop behavior (PHB) that is a mechanism for enforcing QoS for different flows in the IP Backbone. The PacketCable Interdomain Quality of Service Specification [5] defines separate per hop behavior for media flows versus signaling flows. It also provides PHB rules of CMTS managing of upstream bandwidth.

The use of DiffServ Code Points (DSCP) and their associated per hop behavior (PHB) MUST comply with requirements in [5].[IQoS]

7.2.2.2.1 Security Services

Authorization: This is the only security service provided by DiffServ. Based on the DSCP, each IP packet is authorized for a different level of QoS at each DiffServ router. As DiffServ does not provide any authentication, it is possible for an unauthorized router to remark an IP packet with an invalid QoS level. This DiffServ vulnerability is generally tolerated because cryptographic processing on each packet at each router is considered to be too much overhead.

7.2.2.2.2 DiffServ Summary Security Profile Matrix

Table 12. Security Profile Matrix – IQoS

	DiffServ Router- DiffServ Router
authentication	no
access control	Yes, in the form of DSCP authorization
integrity	no
confidentiality	no
non-repudiation	no
Security Mechanisms	*PHB Authorization based on DSCP.* DSCP marking is performed on the upstream channel of edge and border routers.

7.2.2.3 Resource reSerVation Protocol (RSVP)

RSVP is normally used to preserve bandwidth for individual media stream sessions. This type of admission control may be provided at each PacketCable CMTS. When the IP traffic is entering the IP Backbone, keeping track of every individual resource reservation does not scale well enough for the expected growth of PacketCable networks. For this reason, the PacketCable IQoS Specification [5] optionally defines the aggregation of RSVP reservations at the edge and border routers. In general, the IQoS Specification defines various RSVP mechanisms for admission control, but none of them are required for PacketCable. This section specifies the PacketCable profile for RSVP security for those PacketCable elements that employ RSVP.

The aggregation (edge and border) routers have the responsibility of creating aggregate reservations across an aggregation region, which may be the entire DiffServ cloud or a defined aggregation region within the cloud. Each aggregate reservation represents an aggregate flow of traffic from an ingress router (or aggregator) to an egress router (the de-aggregator). Aggregate reservations may be configured statically based on the expected load from an ingress to an egress router, or they may be automatically established and re-sized. Each aggregate reservation carries the traffic from a number of "end-to-end" RSVP reservations that share a common ingress/egress router pair. An end-to-end reservation represents a single microflow, and signaling for such a reservation is accomplished using standard RSVP. "End-to-end" RSVP messages are originated by the CMTS on behalf of the MTA. Such E2E RSVP messages are "tunneled" across the aggregation region by setting the IP protocol number in the Path message to "RSVP-E2E-IGNORE".

7.2.2.3.1 Security Services

Authentication: required to prevent theft of bandwidth and denial of service attacks. Because RSVP messages are modified at each RSVP capable router, RSVP authentication is performed separately at each RSVP hop.

Access Control: required on this interface. The mechanism used to provide access control and policy information at RSVP routers is currently out of scope for PacketCable.

Message Integrity: required on this interface. Without it, the same bandwidth theft scenarios are still possible, along with the denial of service attacks.

Confidentiality: not supported on this interface as it is not part of the IETF standard. Normally, bandwidth reservation signals via RSVP do not carry application level data that needs additional privacy protection.

7.2.2.3.2 Cryptographic Mechanisms

Authentication and Message Integrity: All RSVP messages MUST include an RSVP INTEGRITY object as it is specified in [48]. The RSVP integrity object contains a keyed message digest over the entire RSVP message. Within PacketCable, the keyed message digest algorithm MUST be HMAC-MD5.

In order to support secure RSVP router restart/recovery, all RSVP routers MUST support the integrity handshake mechanism as specified in [48].

Access Control: automatically provided with the use of pre-shared keys. Access is granted to another router when pre-shared keys are established with that router.

7.2.2.3.3 Key Management

Pre-shared keys MUST be used to secure all RSVP interfaces. PacketCable has reviewed RFC 2752, Identity Representation for RSVP where public key and Kerberos approaches are used for RSVP authentication. Within PacketCable, public key and Kerberos approaches for RSVP authentication MUST NOT be used.

Public key authentication is considered to be too computationally expensive to be used for RSVP. Kerberos authentication, as it is specified in RFC 2752, is incomplete.

7.2.2.3.4 RSVP Summary Security Profile Matrix

Table 13. Security Profile Matrix – RSVP

	RSVP Router- RSVP Router
authentication	yes
access control	yes
integrity	yes
confidentiality	no
non-repudiation	no

Security Mechanisms	*HMAC-MD5 for message authentication and integrity. Pre-shared keys satisfy access control requirements.*

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.

7.3.2 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.

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 binary **Element ID** of the CMS or the CMTS. 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.

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

CMS and RKS will negotiate a shared secret (CMS-RKS Secret) using IKE. IKE MUST be use one of the modes with pre-shared keys for this interface. For details, refer to section 6.1.2.3.

IKE MUST run asynchronous to the billing event generation and will guarantee that there is always a valid, non-expired CMS-RKS Secret. This shared secret MUST be unique to this particular CMS and RKS.

At the RKS, the CMS 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 billing event messages. For more details refer to [31].

Later, when a billing event arrives at the RKS, it MUST be able to query the database of IPSec Security Associations and retrieve a source IP address, based on the Element ID. The RKS MUST ensure 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 specified in [42].

7.3.3.2 CMTS – RKS Interface

CMTS and RKS MUST negotiate a shared secret (CMTS-RKS Secret) using IKE. IKE may use one of the modes with pre-shared keys. For details, refer to section 6.1.2.3.

IKE MUST be running asynchronous to the billing event generation and will guarantee that there is always a valid, non-expired CMTS-RKS Secret. This shared secret SHOULD be unique to this particular CMTS and RKS.

At the RKS, the CMTS 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 billing event messages. For more details refer to [31].

Later, when a billing event arrives at the RKS, it MUST be able to query the database of IPSec SAs and retrieve a source IP address, based on the Element ID. The RKS

MUST ensure 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 specified in [42].

7.3.4 CMS – CMTS RADIUS Summary Security Profile Matrix

Table 14. Security Profile Matrix – Radius

	RADIUS Accounting (CMS-Radius Server/RKS)	**RADIUS Accounting (CMTS – Radius Server/RKS)**
authentication	yes	yes
access control	yes	yes
integrity	yes	yes
confidentiality	yes	yes
non-repudiation	no	no
Security mechanisms	*IPSec ESP with encryption and message integrity enabled.* key management using IKE with pre-shared keys	*IPSec ESP with encryption and message integrity enabled* key management using IKE with pre-shared keys

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.

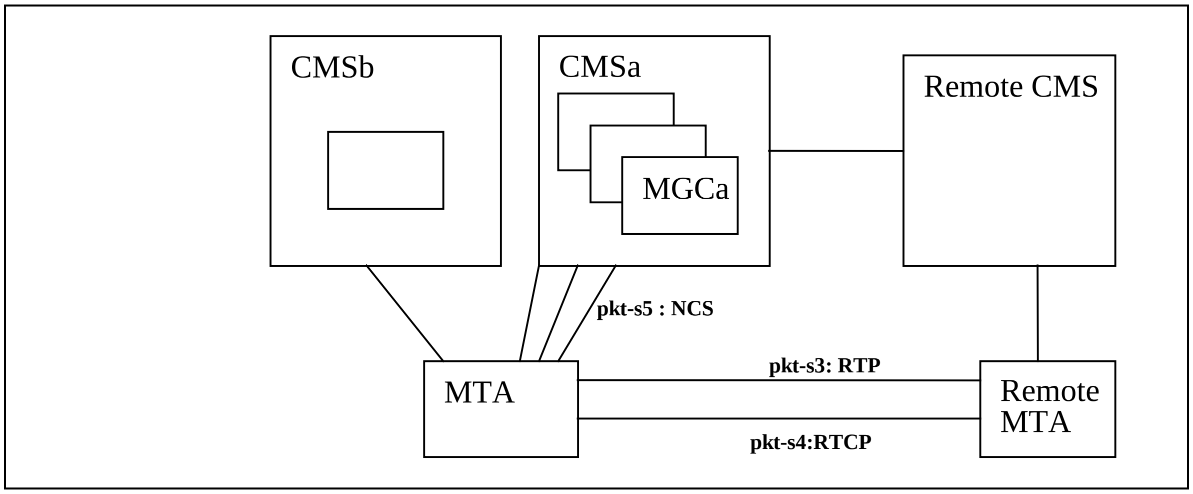

Figure 14. NCS Reference Architecture

The diagram shows the CMSs as being a cluster of several MGCs. 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 [49].

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 [49] 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.

Administratively, various SIP proxies and CMSs are grouped into Kerberos realms. If there is a signaling path between two realms, there is a trust relationship between the corresponding KDCs.

7.4.1.2 Security Services

The same set of requirements applies to both CMS-MTA and CMS-CMS signaling interfaces.

Authentication: signaling messages must be authenticated, in order to prevent a third party masquerading as either an authorized MTA or CMS.

Confidentiality: NCS messages carry dialed numbers and other customer information, which must not be disclosed to a third party. Thus confidentiality of signaling messages is required.

Message integrity: must 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 this interface. IPSec keys MUST be derived using mechanism described in section 6.5.6.

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 <-> 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-SIP Proxy and SIP Proxy – SIP Proxy Interfaces

When a CMS or a SIP Proxy receives a SIP signaling message, it MUST 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 MUST utilize an IP address <-> FQDN map for all CMSs and SIP Proxies that have current IPSec security associations with this host. This map is built during key management described in the following section and does not need to reside in permanent storage.

Whenever the SIP signaling message contains the FQDN of the previous hop (e.g. in the SIP Via header field), that FQDN MUST be identical to the FQDN in the local IP address <-> FQDN map. If they are not the same, the SIP message MUST be rejected as an invalid message.

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 6.4.4). . 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 6.4.5).

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 TGS 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 6.5.6.5.3 describes the technique for the CMS to initiate key management and establish new Security Associations.

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.

7.4.1.4.1.1 Call Agent Clustering

In the case a CMS 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 TGS and multiple AP Request/Reply sequence for each Call Agent in the cluster. The Kerberos messages are specified in section 6.4.5

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 6.5.4).

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:

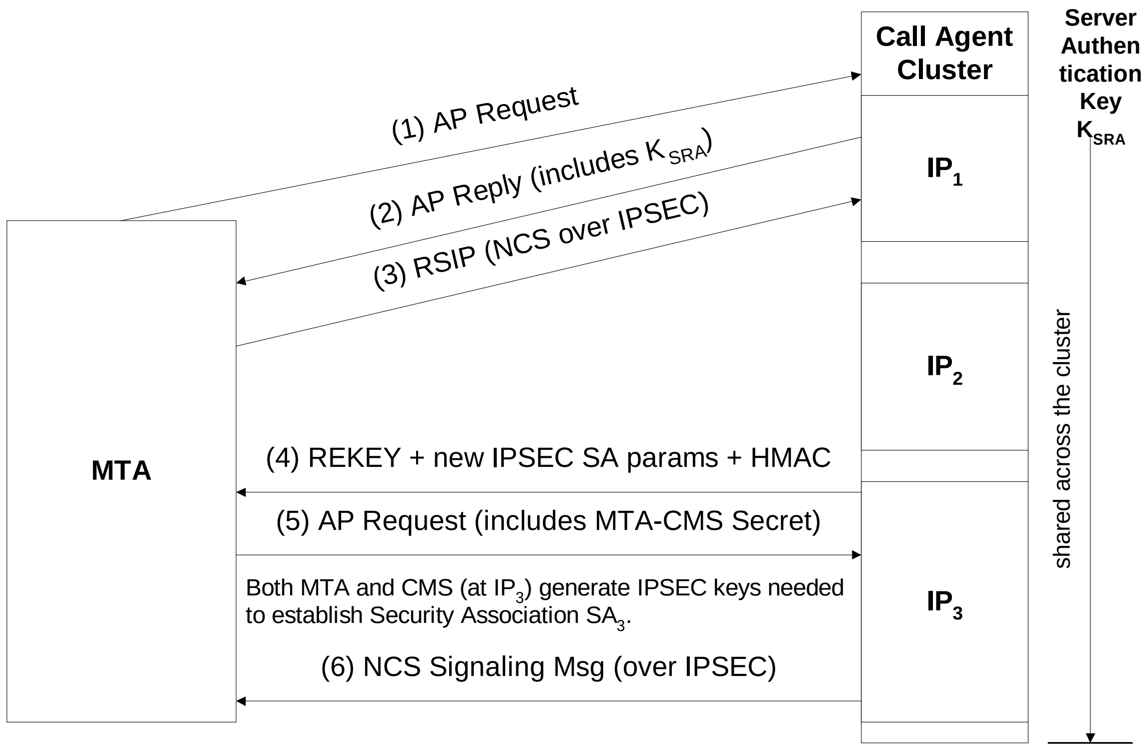
Call Agent Cluster
Server Authen tication Key K_{SRA}
(1) AP Request
(2) AP Reply (includes K_{SRA})
(3) RSIP (NCS over IPSEC)
IP_1
IP_2
IP_3
MTA
(4) REKEY + new IPSEC SA params + HMAC
(5) AP Request (includes MTA-CMS Secret)
Both MTA and CMS (at IP_3) generate IPSEC keys needed to establish Security Association SA_3.
(6) NCS Signaling Msg (over IPSEC)
shared across the cluster

Figure 15: 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 1st 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 IP3 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): ASD, 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 6.5.4.

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 TGS, the MTA will need to execute multiple PKINIT Request/Reply, one for each CMS and then multiple AP Request/Reply in order to create the security association with the individual MGCs.

7.4.1.4.2 CMS-CMS, CMS-SIP Proxy, SIP Proxy-SIP Proxy Key Management

When a CMS or a SIP Proxy has data to send to another CMS or SIP Proxy and doesn't already have Security Associations with that host, it MUST utilize a Kerberized key management protocol (see section 6.5) to establish them. In this case, any CMS or SIP Proxy is responsible for obtaining a ticket for to which it wants to connect; therefore the Wake Up message MUST NOT be used.

A CMS or a SIP Proxy MUST first obtain a TGT (Ticket Granting Ticket) before it can obtain an Application Server ticket for another CMS or SIP Proxy. In order to obtain a TGT, a CMS or a SIP Proxy MUST authenticate itself to the KDC with its symmetric service key (see section 6.4.4). Once a CMS or a SIP Proxy has obtained a TGT, it uses the TGS Request/TGS Reply exchange to obtain the Application Server ticket (see section 6.4.5).

7.4.1.4.2.1 Inter-Domain Call Setup

In the case of an inter-domain calls, some of the signaling hops between the two CMSs may contain two hosts belonging to two different Kerberos realms.

In the cross-realm case, Security Associations in the form of KDC-KDC tickets are used to establish trust between the two KDCs in two different realms. These tickets are obtained with the PKCROSS protocol. After the KDC-KDC trust is established, a CMS or SIP Proxy in one realm is able to get a service ticket for a CMS or SIP Proxy in another realm (see section 6.4.7 for details). After the service ticket is obtained, the Kerberized IPSec key management flows remain the same for the cross-realm case.

7.4.1.4.2.2 Example of Inter-Domain Call Setup with Key Management Flows

A CMS or a SIP Proxy SHOULD create SAs ahead of time (before they are needed) whenever possible. One such example is illustrated in the following diagram.

The diagram illustrates the beginning of an inter-domain call setup. In this example, the signaling between the two CMSs is routed through two EBPs (Exterior Border Proxies). After one roundtrip between the CMSs (after the 183 SDP message is received by CMS "A"), the rest of the CMS-CMS signaling is done directly, without the involvement of the intermediate EBPs.

The diagram assumes that there are no prior SAs between the two CMSs and that CMS "A" does not possess a service ticket for CMS "B". It shows the key management flows necessary to establish the necessary SAs for this call.

CMS "A" sends a TGS Request to its local KDC (KDC "A") to get a ticket for CMS "B". This TGS Request is sent about the same time as the Gate-Set message to the local CMTS. The key management flows continue in parallel with the subsequent DQoS flows (Gate-Set ACK) and also in parallel with some NCS signaling flows (MDCX from CMS "A" to MTA A and the 100 response from MTA A). After the 100 response no more parallelism is possible. The next signaling message that CMS "A" sends out is the PRACK to CMS "B" that has to wait for the SAs with CMS "B" to be set up.

Since CMS "B" is in a different realm (and not in KDC "A's" database), the TGS Request from CMS "A" causes KDC "A" to perform a DNS lookup to retrieve CMS "B's" realm name. After the DNS lookup is complete, KDC "A" will attempt to locate a ticket for KDC "B" in its local ticket cache. If it finds that ticket, it will immediately return to CMS "A" the cross-realm TGT needed to authenticate to KDC "B". If (in a rare circumstance) KDC "A" does not currently have a ticket for KDC "B", it will first have to perform DNS lookups to locate it and then perform a PKCROSS exchange with KDC "B" to obtain the KDC ticket.

Once CMS "A" finally receives a cross-realm TGT for KDC "B", it has to send another TGS Request for KDC "B" and then finally obtain the service ticket for CMS "B". In order for CMS "A" to contact KDC "B", it will first have to perform two more DNS queries (one to get the KDC "B's" FQDN and another to get its IP address).

After CMS "A" had obtained a ticket for CMS "B", it will initiate Kerberized IPSec key management with CMS "B" in order to set up SAs. The PRACK message from CMS "A" to CMS "B" will be secured with these newly set up SAs.

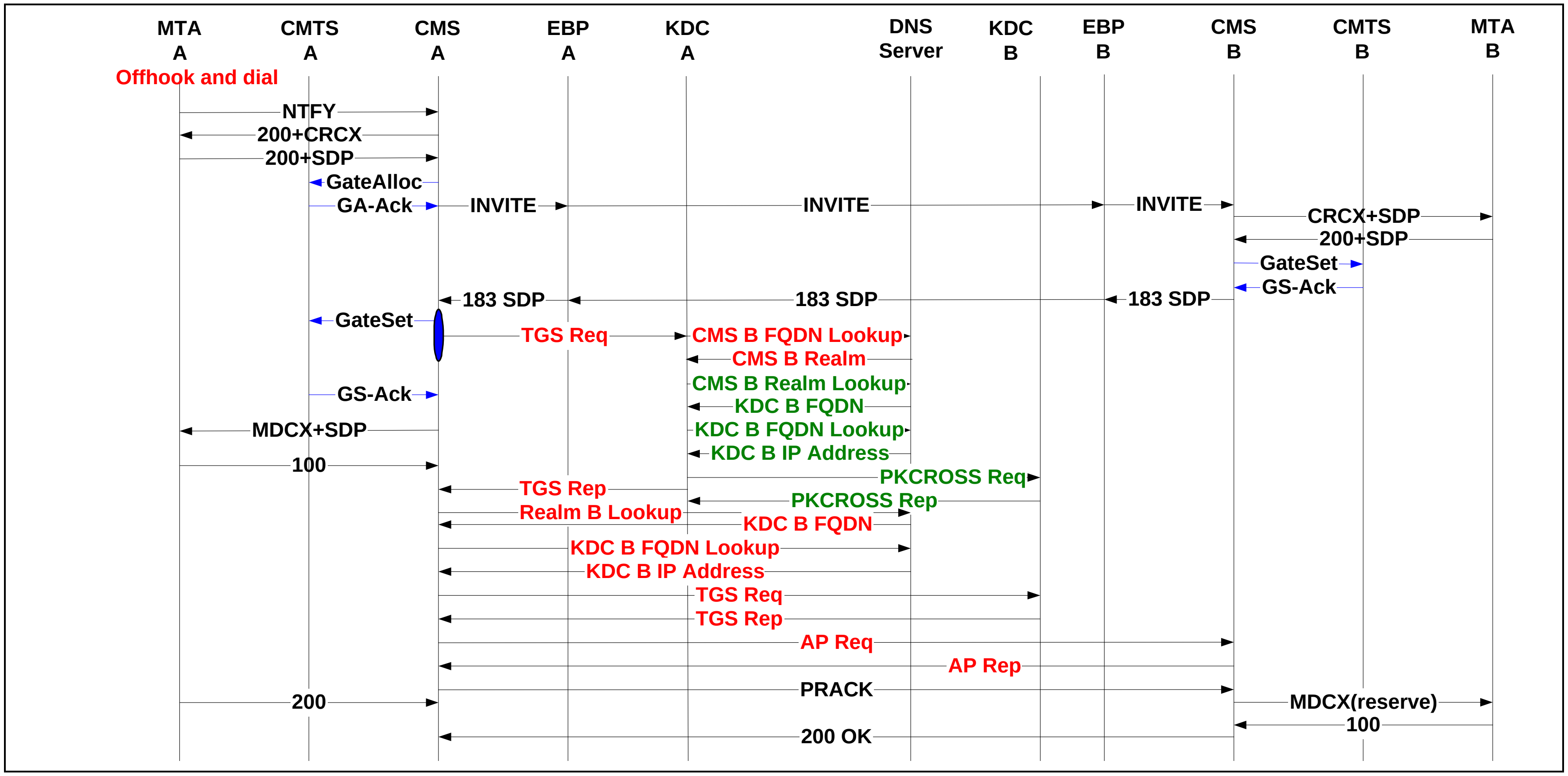

Figure 16. CMS – CMS Signaling Flow with Security

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 [7] 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.

7.5.2.2 MGC – SG Interface

Authentication: signaling messages must be authenticated, in order to prevent a third party masquerading as either an authorized MGC or SG.

Access Control: Services enable by the NCS signaling should be made available only to authorized users – thus access control is required at the MGC.

Integrity: must be assured in order to prevent tampering with the signaling messages – e.g. changing the dialed numbers.

Confidentiality: NCS messages carry dialed numbers and other customer information, which must not be disclosed to a third party. Thus confidentiality of signaling messages is required.

7.5.2.3 CMS – SG Interface

This interface is used for TCAP queries for LNP (Local Number Portability) and other voice communications services.

Authentication: TCAP queries must be authenticated, in order to prevent release of information to an unauthorized party.

Access Control: required along with the authentication, in order to prevent release of information to an unauthorized party.

Integrity: must be assured in order to prevent tampering with the TCAP queries, to prevent a class of denial of service attacks.

Confidentiality: TCAP queries contain dialed numbers and other subscriber information that MUST be kept private. Thus, confidentiality 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 vise versa. Refer to section 6.1 for details of how IPSec ESP is used within PacketCable and for the list of available ciphersuites.

The ISTP protocol allows multiple redundant connections between the SG and the MGC. Multiple connections mean multiple security associations. The assumption is that the number of multiple connections is manageably small, where ahead of time we would set up a security association for each one, using IKE with pre-shared keys.

7.5.3.2 MGC – SG Interface

IPSec ESP MUST be used to both authenticate and encrypt the messages from MGC to SG and vise versa. Refer to section 6.1 for details of how IPSec ESP is used within PacketCable and for the list of available ciphersuites.

7.5.3.3 CMS – SG Interface

This interface is used for TCAP queries for LNP (Local Number Portability) and other voice communications services. IPSec ESP MUST be used to both authenticate and encrypt the messages from CMS to SG and vice versa. Refer to section 6.1 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 MGC – MG interface MUST be implemented via IKE. IKE MUST use pre-shared key mode for this interface. Refer to section 6.2 of this document for details on the PacketCable use of IKE.

IKE will guarantee that there is always a valid, non-expired MGC – MG Secret. This shared secret MUST be unique to this particular interface.

7.5.4.2 MGC – SG Interface

Key management for MGC – SG interface MUST be implemented via IKE. IKE MUST use pre-shared key mode for this interface. Refer to section 6.2of this document for details on the PacketCable use of IKE.

IKE will guarantee that there is always a valid, non-expired MGC – SG Secret. This shared secret MUST be unique to this particular interface.

7.5.4.3 CMS – SG Interface

Key management for CMS – SG interface MUST be implemented via IKE. IKE MUST use pre-shared key mode for this interface. Refer to section 6.2 of this document for details on the PacketCable use of IKE.

IKE will guarantee that there is always a valid, non-expired CMS – SG Secret. This shared secret MUST be unique to this particular interface.

7.5.5 MGC-MG-CMS-SG Summary Security Profile Matrix

Table 15. Security Profile Matrix – TCAP/IP & TGCP

	TCAP-IP , ISUP-IP (MGC-SG)	TGCP (MG – MGC)	TCAP-IP (CMS-SG)
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* IKE with pre-shared keys	*IPSec* IKE with pre-shared keys	*IPSec* IKE with pre-shared keys

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 must 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 7.6.2.1.2.1).

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: some RTCP messages (e.g. CNAME) contain the identity of the endpoint. The requirement is to keep all RTCP messages private, so that for simplicity, encryption can be done below the application layer (with IPSec).

Message Integrity: RTCP signaling messages (e.g. BYE) can be manipulated to cause denial of service attacks. To prevent these attacks, message integrity is required for RTCP.

7.6.2 Cryptographic Mechanisms

Each media RTP packet MUST be encrypted for privacy. The MTAs have an ability to negotiate a particular encryption algorithm. Encryption MUST be applied to the packet's payload. Encryption MUST NOT be applied to its header.

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 7.6.2.2.

7.6.2.1 RTP Packet Format

Figure 18 shows the format of an encoded RTP packet. PacketCable MUST adhere to the RTP packet format as defined by RFC 1889 and RFC 1890.

The packet's header consists of 12 or more octets, as described in [14]. 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):

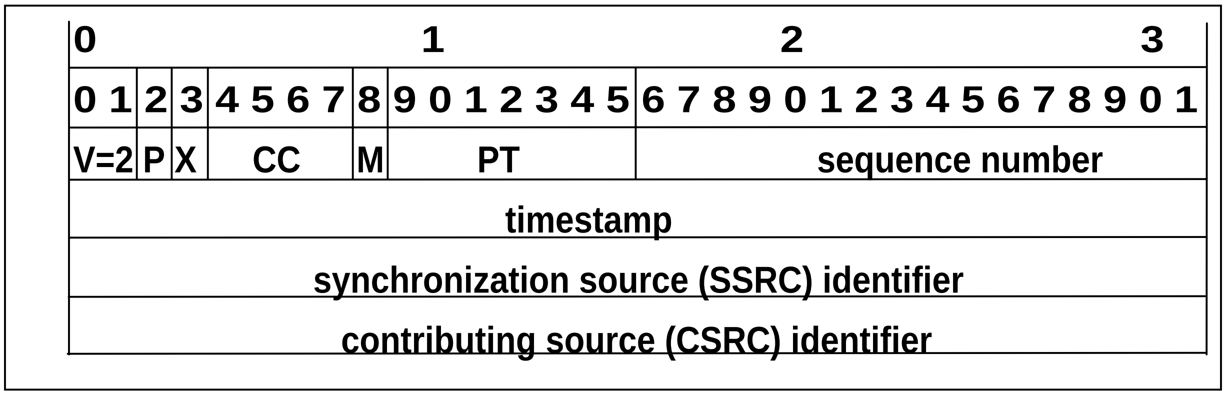

Figure 17. RTP Packet Header Format

The first twelve octets are present in every RTP packet, while the list of CSRC identifiers is present only when inserted by a mixer.

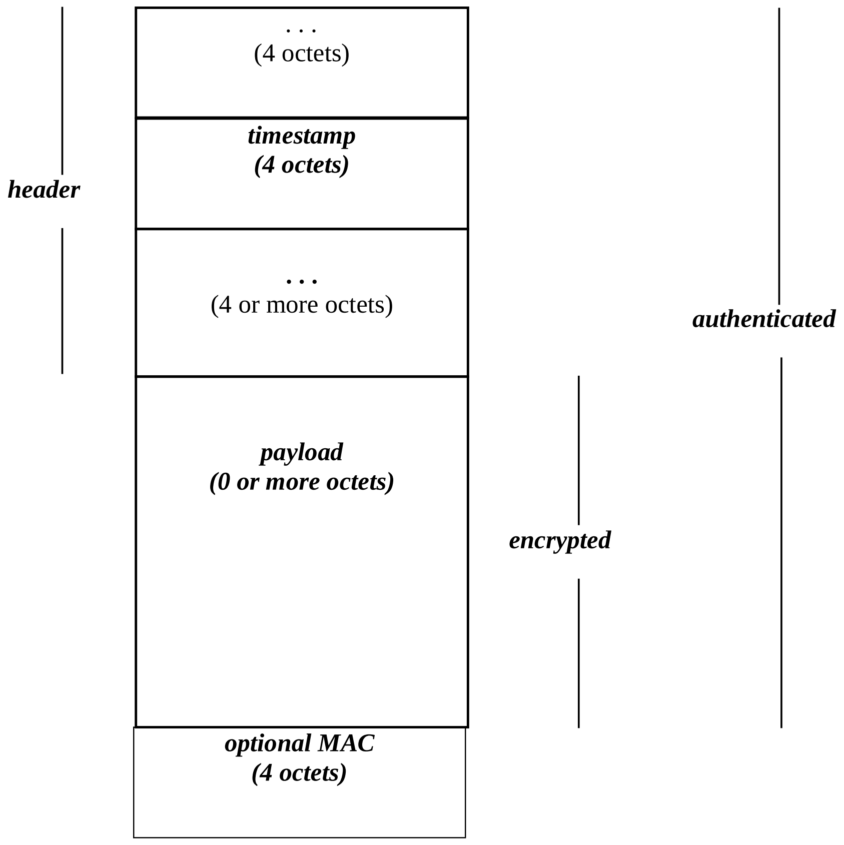

Figure 18. Format of Encoded RTP Packet

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 in an event packet within the duration of one codec frame. The maximum size of the payload of an event packet is M_f*N_e.

  Note: PacketCable will use $N_e = N_c$, i.e., an event packet can have a payload as large as that of a voice packet, but no longer. Future versions of PacketCable may provide a means for N_e to be determined in other ways.
- 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.

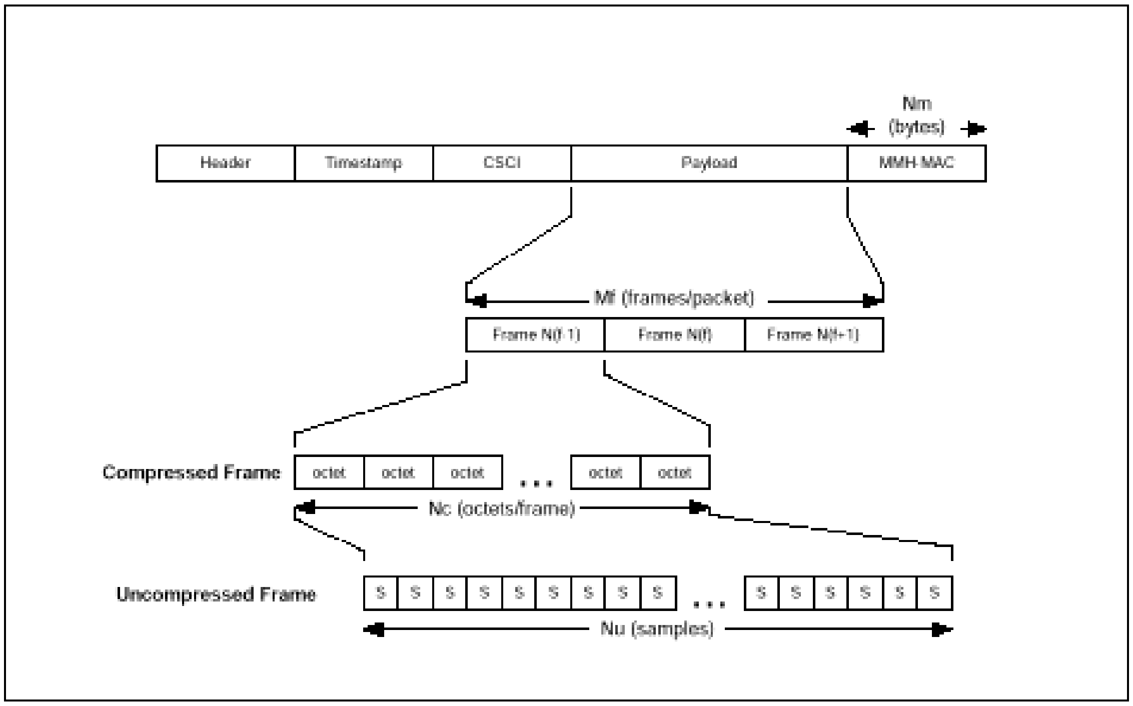

Figure 19. RTP Packet Profile Characteristics

7.6.2.1.1 RTP Timestamp

According to RFC1889, the timestamp field is a 32-bit value initially chosen at random. According 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 7.6.2.2.3.

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 RC4 Encryption and MMH MAC Option

7.6.2.1.2.1.1 Deriving an MMH MAC Key

Key size is (N_h + N_e + N_m –1), where: N_h is the maximum number of octets in the RTP header; N_e is maximum number of octets in an event packet's payload, as defined in section 7.6.2.1; and N_m is the number of octets in the MAC. The number of octets in the RTP header ranges from 12 to 72, inclusive, depending on the number of CSRC identifiers that are included [14]. 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.

7.6.2.1.2.1.2 Initializing the RC4 Encryption Process

The following additional parameter is defined for use with RC4:

- N_k, the state of the RC4 encryption process. "State" means the number of keystream octets that have been previously generated by the process, whether used or discarded. N_k has value 0 immediately after the RC4 process is initialized with a new key and increments with each generated octet of keystream.

  Prior to encoding the first packet, the following procedure MUST be used:

- The new RTP Privacy Key is used to initialize the RC4 encryption process.
- N_k is initialized to 0.
- N_f is initialized to 0.

7.6.2.1.2.1.3 Packet Encoding

Each packet MUST be encoded using the following procedure:

- The timestamp is written into the timestamp field of the header. The timestamp MUST equate to the value: ((N_f*N_u) + (RTP Initial Timestamp)) modulo 2^{32}, where N_f is the frame number of the first frame included in the packet.
- All other fields of the header are set to values prescribed in [15].
- The RC4 encryption state N_k is set to the value N_f*(N_e + N_m).
- The octets of the packet's payload are encrypted using the RC4 encryption process and inserted into the payload field. If there are B octets to be encrypted, then they are encrypted using octets N_k+N_m to N_k+N_m+B-1, inclusive and in order in the RC4 keystream.
- If the MAC option is enabled, the MAC digest is computed using the MMH algorithm with the RTP MAC Key. The digest calculation begins with the first octet of the unencrypted header and ends with the last octet of the encrypted

payload. The computed digest is inserted into the MAC field. The digest calculation requires N_m octets of keystream from the RC4 process. These N_m octets are taken from the octets N_k to N_k+N_m-1, inclusive and in order in the RC4 keystream.

Not all of the keystream octets generated by the RC4 process are necessarily used. If a packet contains m frames, then the RC4 state is advanced by $m*(N_e + N_m)$ prior to encoding the next packet. However, only $m*N_c + N_m$ keystream octets are actually used to encode the current packet; the remaining $(m - 1)*N_m + m*(N_e - N_c)$ keystream octets are unused. The RC4 encryption process is advanced for silent codec frames that are not actually transmitted, since the value of N_f increments even for silent frames. For each dropped silent frame, $(N_e + N_m)$ keystream octets are unused. Instead of dropping a silent frame, a codec might encode it using a short frame containing s octets, where $s < N_c$. For such a short frame, $(N_e - s)$ keystream octets are unused.

7.6.2.1.2.1.4 Codec change

If the codec is changed without changing the keying material, the sender must continue its RC4 encryption process from its state prior to the codec change. It cannot reset the state to 0, since a stream cipher such as RC4 may not reuse keystream without exposing plaintext. The sender also needs to preserve continuity of the timestamp across the codec change, since the timestamp reflects real time.

Changing the codec is likely to change the frame parameters. This changes the proportionality constant that relates the RC4 state to the timestamp. To preserve continuity of the RC4 state and the timestamp across the codec change, the sender MUST compute a new frame number. The new frame number is applied to the first frame generated by the new codec.

The new frame number MUST be calculated as follows:

$$N_{f,new} = \text{roof}\left((N_{f,old} + 1)* (N_{e,old} + N_m) / (N_{e,new} + N_m)\right)$$

where: $N_{f,new}$ is the new frame number; $N_{f,old}$ is the frame number of the last frame generated by the old codec, whether transmitted or dropped as silent; $N_{e,old}$ and $N_{e,new}$ are the values of N_e under the old and new codecs, respectively; and roof(x) is the function that returns the smallest integer no less than x. The new frame number is applied to the first frame generated by the new codec; thereafter, the frame number is incremented by 1.

To encode the first packet containing frames of the new codec, the sender will, in general, need to advance its RC4 state slightly from its state after processing the last packet under the old codec. This is because of rounding error introduced by requiring the frame number to be an integer value. The amount by which the RC4 state needs to be advanced is less than $(N_{e,new} + N_m)$.

7.6.2.1.2.1.5 Change in the MAC Size

As explained in section 7.6.2.1.3.3, the MTA that initiated a call has to be prepared to receive RTP packets before ciphersuite negotiation has been completed. Once the negotiation is complete, it may turn out that the negotiated ciphersuite uses a MAC of a different size than what the initiating MTA assumed previously. At that point, the MTA will have to re-adjust the frame number based on the new MAC size.

This procedure of re-adjusting the frame number is almost identical to the one described in section 7.6.2.1.2.1.4above for codec changes. The only difference, is that it is the MAC size N_m and not the codec that changes:

$N_{f,new} = \text{roof}\ ((N_{f,old} + 1) * (N_e + N_{m,old}) / (N_e + N_{m,new}))$

Here, $N_{m,old}$ and $N_{m,new}$ are the old and new MAC sizes respectively.

7.6.2.1.2.2 Block Cipher Encryption of RTP Packets

7.6.2.1.2.2.1 Block Termination

If an implementation supports block ciphers, the residual block termination (RBT) MUST be used to terminate streams that end with less than a full block of data to encrypt. (see section 9.6.2.2).

7.6.2.1.2.2.2 Initialization Vector

An 8-byte initialization value (IV) is required when using 8-byte block ciphers in CBC mode to encrypt RTP packet payloads. The IV MUST be calculated new for each RTP packet using the ECB mode of the same block encryption algorithm defined for encrypting the payload. The first 64-bits of each RTP packet header (including sequence-number and timestamp) are first X-OR'ed with a 64-bit secret value. The secret value is calculated from the End-to-End secret according to 4.7.3.2.3. The result of the X-OR is encrypted by the block cipher using the same key(s) that will be used to encrypt the RTP payload. The resulting ciphertext will be the 64-bit IV.

7.6.2.1.2.2.3 MMH-MAC Pad Derivation When Using a Block Cipher

A method for deriving the MMH-MAC pad when using RC4 was explained in an earlier section. When using a block cipher, the pad is calculated using the block cipher. When the block cipher requires an IV, the IV value is calculated according to section 9.6.2.27.6.2.1.2.2.2. This value will serve as the basis of the MMH-MAC pad when using a block cipher. If the MMH-MAC is used with a block cipher that does not require an IV, a corresponding value MUST be calculated according to section 9.6.2.27.6.2.1.2.2.2 and used as the basis of the MMH-MAC pad according to this section.

The pad is subsequently calculated by performing the MMH digest function on the resulting IV and then using the appropriate number of most-significant-bytes for the MMH-MAC pad.

Additional keying material needed to calculate the digest for the pad is derived with the MMH-MAC key from the shared secret as specified in section 7.6.2.1.2.2.3.

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 7.6.2.2.3:

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 the items (1) through (4) 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.

3) If the timestamp value in the RTP headers from a sender never comes back within the acceptable range, the receiver discontinues the session.

4) 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.

7.6.2.1.3.3 RC4 Decryption and MMH MAC

Prior to decoding the first packet, the RTP Privacy Key is used to initialize the RC4 decryption process state N_k to zero.

Each packet MUST be decoded using the following procedure:

- The frame number for the first frame in the packet, N_f, is computed from the value of the timestamp field in the header as follows:

    - if the value of the timestamp is greater than or equal to the value of RTP Initial Timestamp, then N_f = (timestamp – (RTP Initial Timestamp)) / N_u;
    - otherwise, N_f = (timestamp + $2^{32\,-}$ (RTP Initial Timestamp) / N_u.
- If the computed value of N_f is not an integer value, the packet is discarded; this indicates an invalid timestamp.
- The RC4 decryption state N_k is set to the value $N_f*(N_e + N_m)$.
- If the MAC option is enabled, a MAC digest is computed using the MMH algorithm with the RTP MAC key. The digest calculation begins with the first octet of the unencrypted header and ends with the last octet of the encrypted payload. The digest calculation requires N_m octets of keystream from the RC4 process. These N_m octets are taken from the octets N_k to N_k+N_m-1, inclusive and in order in the RC4 keystream. The computed digest is compared to the value in the MAC field. If the computed digest does not match the value in the MAC field, the packet is discarded.
- The octets of the packet's payload are decrypted using the RC4 decryption process. If there are B octets to be decrypted, then they are decrypted using octets N_k+N_m to N_k+N_m+B-1, inclusive and in order in the RC4 keystream.
- All other fields of the header are processed as prescribed in [15].

Note that the state of the RC4 decryption process is adjusted to match the state of the sender's RC4 encryption process prior to decrypting the packet's payload or verifying its MAC digest. If packets arrive out of order, the receive must, in principle, push the RC4 process backwards, as well as forwards, in order to match the state of the sender's RC4 process. In practice, this can be accomplished by having the receiver run its RC4 process in the forward direction only and synchronized to real time, thus making keystream available to decode packets in whatever order they arrive.

7.6.2.2 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.

At the start of each voice communication, the CMS in the source MTA's domain MUST securely distribute a secret to both MTAs. This secret may be used by both MTAs to derive the keying material for both the bearer channel and the MTA-MTA signaling messages.

The MTAs may choose to ignore this secret distributed by the CMS and negotiate their own keys. However, direct MTA-MTA key negotiation is outside of the scope of PacketCable.

The distribution of End-End Secret is specific to the call signaling described in the following subsections.

7.6.2.2.1 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. However, NCS has not yet defined the CMS-CMS protocol and SIP+ is only one possible choice.

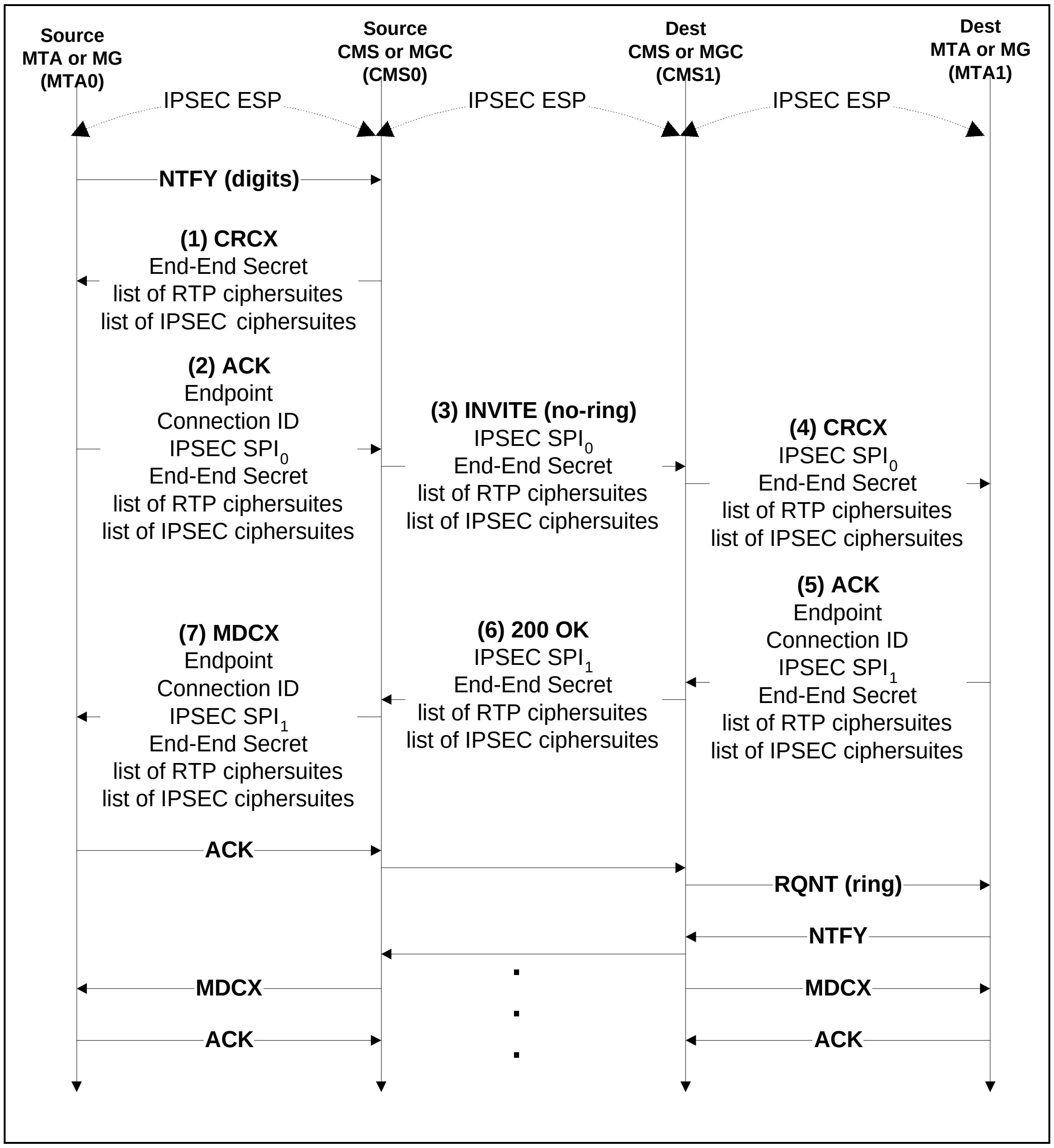

Figure 20. End-End Secret Distribution over NCS

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. Similarly, there is a pre-existing IPSec ESP session between the pair of CMSs. This allows the End-End Secret to be distributed securely, with privacy, integrity and anti-replay mechanisms already in place.

Each of the numbered flows in the above diagram is described below:

(1) CMS0 -> MTA0

CMS0 sends to the MTA0 the allowable lists of ciphersuites for the new communication – one list for RTP security and one for RTCP (over IPSec ESP). It

SHOULD also generate an End-End Secret to be used for this communication and send it to the MTA0 along with the ciphersuites. If CMS0 chooses not to generate the secret, it will be up to MTA0 to do it in the subsequent step. This information is sent with a **CreateConnection** (CRCX) command, inside the **LocalConnectionOptions** parameter.

(2) MTA0 -> CMS0

MTA0 generates Connection ID for this particular connection. Thus, the End-End Secret is now associated with a Connection ID. The pair (Connection ID, Endpoint) uniquely identifies this connection, where the Endpoint is an NCS identifier for MTA0.

From the list of ciphersuites in the **LocalConnectionOptions**, MTA0 MUST select a (non-empty) subset for RTP and a (non-empty) subset for RTCP and returns them in the subsequent ACK message, in the **LocalConnectionDescriptor**, in the form of SDP attributes. The resulting lists of ciphersuites are in a prioritized order – the preferred ciphersuites MUST be listed first.

MTA0 SHOULD take the End-End Secret from the **LocalConnectionOptions** and return it along with the ciphersuites in the **LocalConnectionDescriptor**. MTA0 MAY also generate a new End-End Secret and include it in the **LocalConnectionDescriptor** instead. In the case that CMS0 did not send the End-End Secret, MTA0 itself MUST generate the secret.

The ACK also includes the Connection ID and the Endpoint for MTA0.

In order to receive RTCP packets over IPSec, MTA0 generates a locally unique 4-byte identifier, called ASD (Application-Specific Data). This ASD (ASD_0) MUST also be included in the **LocalConnectionDescriptor**, used to identify the SA for the incoming RTCP packets.

Right after sending this message, MTA0 SHOULD establish its inbound RTP and RTCP (IPSec ESP) SAs, based on the first (preferred) ciphersuite in the RTP and RTCP lists. MTA0 SHOULD be ready to receive RTP and RTCP messages, which may arrive any time after this message is received by the CMS.

If later MTA1 decides to use alternate ciphersuites listed by MTA0, MTA0 will later have to update its RTP and RTCP Security Associations. If MTA1 also decides to send MTA0 packets before ciphersuite negotiation had completed, processing on those packets at MTA0 will fail (since it assumed a different ciphersuite).

(3) CMS0 -> CMS1

The CMS0 MUST send the End-End Secret and a list of ciphersuites selected by MTA0 to MTA1's local CMS (CMS1). CMS1 will later forward this information to MTA1. Although this message is shown in the diagram as a SIP+ **INVITE**, the CMS-CMS protocol has not yet been specified by NCS.

(4) CMS1 -> MTA1

CMS1 MUST relay the same End-End Secret (generated by CMS0 or MTA0) for the incoming communication to MTA1 along with the lists of ciphersuites selected by MTA0 – one list for RTP security and one for RTCP (over IPSec ESP). This information MUST be sent with a **CreateConnection** (CRCX) command, inside the **LocalConnectionOptions** parameter. (It MAY also be present in the **RemoteConnectionDescriptor** parameter in the form of SDP attributes).

(5) MTA1 -> CMS1

MTA1 MUST generate Connection ID for this particular connection. Thus, the End-End Secret is now associated with a pair (Endpoint, Connection ID).

From the list of ciphersuites in the **LocalConnectionOption**, MTA1 SHOULD select the first listed ciphersuite for RTP and the first one for RTCP. This allows MTA1 to immediately start sending RTP and RTCP packets to MTA0. MTA1 MAY also select alternate ciphersuites specified by MTA0. In that case, MTA1 SHOULD not try to send any packets to MTA0 until it is certain that MTA0 had been informed of the selected ciphersuites – after receiving a message labeled RQNT (ring) in the above diagram.

MTA1 MUST send back an ACK message, which includes lists of the selected ciphersuites inside the **LocalConnectionDescriptor**, in the form of SDP attributes. The 1st ciphersuite in each list (one for RTP and one for RTCP) MUST be the one that was already selected by MTA1. Additional ciphersuites in each list are alternatives. If at any time, MTA0 wants to switch to one of the alternatives that were selected by MTA1, it would have to go through a new key negotiation. The ACK MUST include the Connection ID.

In order to receive RTCP packets over IPSec, MTA1 generates a locally unique 4-byte identifier, ASD_1, used to identify the SA for the incoming RTCP packets, and MUST include it in the **LocalConnectionDescriptor**.

The SDP attributes MUST also include the End-End Secret, even though the MTA1 does not have an option of changing it. The End-End Secret is sent for consistency and to verify that MTA1 is using the right key.

At this time, MTA1 creates both RTP and RTCP (IPSec ESP) SAs, for both inbound and outbound messages, and is ready to send and receive RTP and RTCP messages for this communication.

(6) CMS1 -> CMS0

CMS1 MUST forward the acknowledgement and the selected ciphersuites from MTA1 to CMS0.

(7) CMS0 -> MTA0

CMS0 MUST now send a **ModifyConnection** command to MTA0. If MTA1 chose a ciphersuite that is different from MTA0's preferred choice (sent in step 2), the ciphersuites (and alternatives) MUST be included in the **LocalConnectionOptions**.

MTA0 MUST check if the first RTP and RTCP ciphersuites in the **LocalConnectionOptions** differ from the ones that it selected in step (2). If either ciphersuite had been changed, MTA0 MUST update the corresponding inbound SA. For updating RTP key stream when the MAC size had changed, refer to section 7.6.2.1.2.1.5.

RemoteConnectionDescriptor MAY also include the ciphersuites (and alternatives) selected by MTA1, as well as the End-End Secret, which MAY be used to verify that MTA1 is using the right key.

At this time, MTA0 MUST also establish its outbound RTP and RTCP (IPSec ESP) SAs. MTA0 is now ready to send (in addition to receiving) RTP and RTCP messages to MTA1.

For full syntax of the NCS messages, including the End-End Secret, list of ciphersuites, and other related information, please refer to the NCS signaling specification [2].

7.6.2.2.2 Ciphersuite Format

Each ciphersuite for both bearer channel and signaling security (via IPSec) 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 6.1 for IPSec, and to section 6.6 for the RTP 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.

7.6.2.2.3 Derivation of End-to-End Keys

7.6.2.2.3.1 Initial Key Derivation

After the receipt of the End-End Secret, each MTA MUST independently derive the keys it needs to secure all MTA-MTA communication. The size of the End-End Secret is 46 bytes (the same as with the SSL or TLS pre-master secret).

All keys are derived in pairs, since each individual authentication or encryption key is applied to a unidirectional flow. The keys MUST be derived as follows, in the specified order:

(1) RTP (bearer channel security). Derive two sets of the following keys, one for sending packets and one for receiving. The derivation function is **F(S, "End-End RTP Security Association")**. Here, S is the End-End Secret and the string "End-End

RTP SA" is taken without quotes and without a terminating null character. F is defined in section 9.5.

Appendix A. RTP privacy key. See encryption algorithm options in section 6.6

Appendix B. 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) a 64-bit value used to derive the 64-bit IV according to 7.6.2.1.2.2.2. 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.

(2) IPSec ESP (for RTCP SAs). Derive two sets of the following keys, one for sending packets and one for receiving. The derivation function is F(S, "End-End RTP Control Protocol Security Association").

a) Message authentication key.

b) Encryption key.

For possible message authentication and encryption algorithms that can be used with IPSec, and for the corresponding key sizes, refer to section 6.1.

For each set of keys and security parameters specified above, the MTA (or MG) that initiated the communication MUST derive send parameters first and receive parameters second. The target MTA (or MG) that receives the communication MUST derive receive parameters first and send parameters second.

It is also possible that a MTA is communicating to a regular phone, via a POTS Signaling Gateway. In that case, the scenario is almost identical, except that the destination MTA is replaced with the POTS Signaling Gateway.

Similarly, it is possible for a regular phone to call up a MTA, via a POTS Signaling Gateway. Again, the scenario is almost identical, except that the source MTA is replaced with the POTS Signaling Gateway.

7.6.2.2.3.2 End-to-End Rekey Derivation

When key parameters need to be changed for a current RTP session, they MUST be generated independently by the two endpoints. The derivation function is F(S, "End-End RTP Key Change <N>"). Here, S is the same End-End Secret used to derive the original set of keys for this RTP connection. "End-End RTP Key Change <N>" is taken without quotes and without a terminating NULL character.

The value <N> stands for a counter which will have a value 1 for the 1st key change, then 2, 3, etc. The function F is defined in section 6.6.

The RTCP keys defined in section 7.6.2.2.1 part (2) are not derived during this key change.

The key change derivation MUST occur before the timestamp value rolls-over to a value equal-to or past its initial value from the previous key derivation. The new key parameters MUST be ready ahead of the time they are needed, and not used until the timestamp value rolls-over to a value equal-to or past its initial value. Both the sender and the received MUST maintain both old and new sets of parameters during the rollover transition of the timestamp. Once a sender begins using the new key set, it MUST NOT use the old keys again for outgoing packets. Once the timestamp value requiring the old key parameter set is outside the acceptable tolerance for incoming timestamps (see section 7.6.2.1.1), the receiver MUST delete its old key parameters.

7.6.2.3 RTP-RTCP Summary Security Profile Matrix

Table 16. Security Profile Matrix – RTP & RTCP

	RTP (MTA – MTA, MTA – MG)	**RTCP (MTA – MTA, MTA – MG, MG – MG)**
authentication	yes (indirect)[7]	yes (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* End-to-End Secret distributed over secured MTA-CMS links. Final keys derived from this secret. RC4-128 encryption algorithm Optional 2-byte or 4-byte MAC based on MMH algorithm PacketCable requires support for ciphersuite negotiation.	*IPSec ESP in transport mode with both encryption and message integrity enabled. The UDP check sum may need to be turned off (set to 0), depending on how NAT is implemented.* Keys derived from the same End-to-End Secret as the one used to derive RTP keys.

[7] MTAs do not authenticate directly. Authentication refers to the authentication of identity.

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.

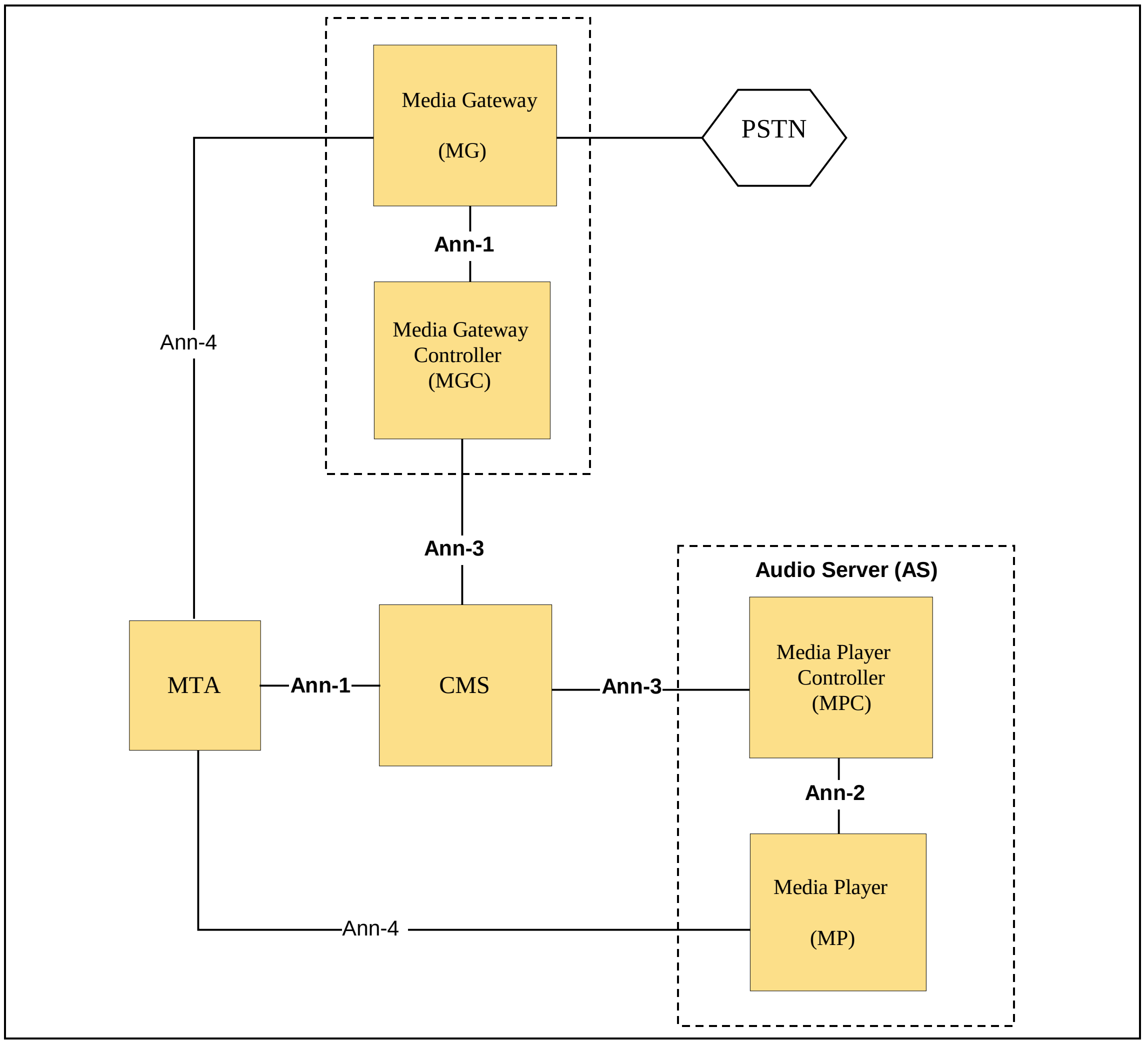

Figure 21. Audio Server Components and Interfaces

Figure 21 shows a network-based Media Player (MP). It has an optional TGCP 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.

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 TGCP 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.

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 vise versa. Refer to section 6.1for details of how IPSec ESP is used within PacketCable and for the list of available ciphersuites.

The MPC MUST verify that the MP domain name (included in the **Endpoint ID**) correctly corresponds to its IP address for each signaling message received from the MP. This check is done via a lookup into a map of IP addresses to MP domain names. Also, in reverse, when the MPC performs a lookup of the MP IP address based on its domain name (in order to send a message to the MP), it consults this same map instead of performing a DNS query. Refer to section 5.1.3.8 on how this map is maintained.

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 7.4.1.4.1.

7.7.4.2 MPC-MP Signaling (Ann-2)

MPC and MP MUST negotiate a shared secret (MPC-MP Secret) using IKE. IKE MUST use one of the modes with pre-shared keys for this interface. For details, refer to section 6.1.2.3.

IKE MUST be running asynchronous to the signaling messages and will guarantee that there is always a valid, non-expired MPC-MP Secret. This shared secret MUST be unique to this particular MPC and MP.

At the MPC, MP domain names MUST be associated with the corresponding IP addresses. One possibility is to associate each pre-shared key directly with the domain name. 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 MUST be the MP domain name. For more details refer to [34].

Later, when a TGCP signaling message arrives at the MPC, it MUST be able to query the database of IPSec Security Associations and retrieve a source IP address, based on the MP domain name. The MPC MUST ensure 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 specified in [42].

Similarly, rather than doing a DNS query to find an IP address of the correct MP, MPC will look it up in the database of IPSec SAs.

7.7.4.3 MTA-MP (Ann-4)

Key Management on the media packet interface is specified in section 7.6.2.2. 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 17. Security Profile Matrix – Audio Server Services

	Ann-1: NCS (MTA – CMS) & (MG – MGC)	**Ann-2: TGCP (MPC-MP)**	**Ann-3: unspecified (CMS-MPC) & (CMS – MGC) Interface Security Requirements***	**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	IPSec IKE with pre-shared keys		Application Layer Security via RTP PacketCable Security Profile keys distributed over secured MTA-CMS and MP-MPC links RC4-128 encryption algorithm Optional 2-byte or 4-byte MAC based on MMH algorithm.	IPSec ESP in transport mode with both encryption and message integrity enabled. Keys derived from bearer channel RTP SA.

*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 consists of the following elements and interfaces:

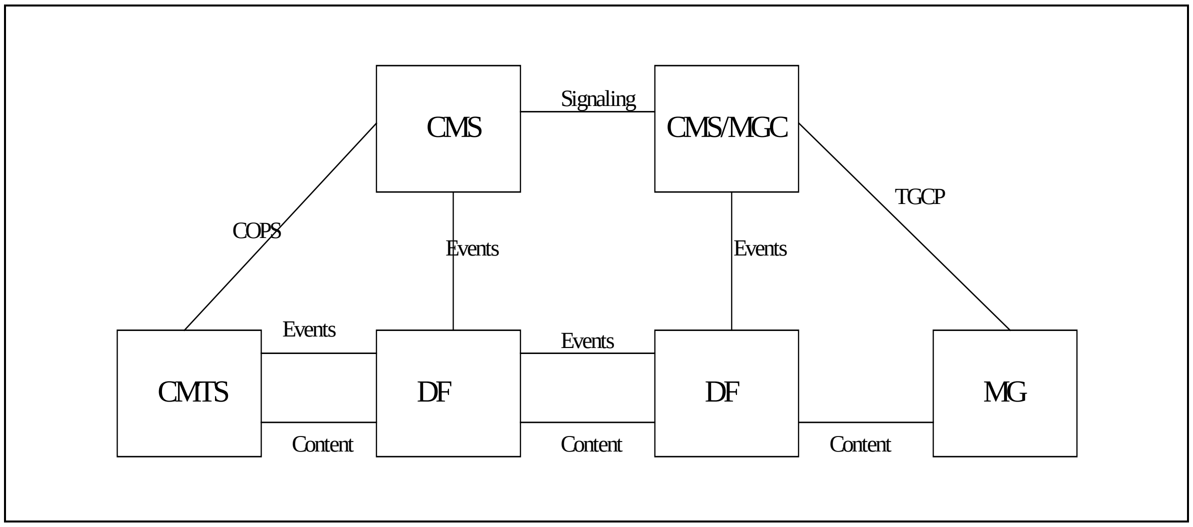

Figure 22. Electronic Surveillance Security Interfaces

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 and the DF provides descriptions of calls, 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 7.2.1.2.3, 7.2.1.3.3and 7.2.1.4.2.

The TGCP signaling interface between the CMS/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, 7.8.3.1and 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.

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, 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 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.

Also the same 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 0-bytes.

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 7.3.1.2.

Also the same 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 0-bytes.

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.

Also the same 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 0-bytes.

7.8.4 Key Management

7.8.4.1 Interface between CMS and DF

CMS and 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. The corresponding IPSec keys MUST be unique to this particular CMS and DF.

At the DF, CMS 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.

7.8.4.2 Interface between CMTS and DF

CMTS and DF MUST negotiate a pair of 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. The corresponding IPSec keys MUST be unique to this particular CMTS and DF.

At the DF, CMTS 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.

7.8.4.3 Interface between DF and DF

CMTS and DF MUST negotiate a shared secret (CMTS-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.

8 PACKETCABLE CERTIFICATES

PacketCable uses digital certificates, which comply with the X.509 specification [51] and the IETF PKIX specification [52].

8.1 Generic Structure

8.1.1 Version

The Version of the certificates MUST be V3. All certificates MUST comply with RFC 2459 [52] except where the non-compliance with the RFC is explicitly stated in the chapter 8 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 modulus for all RSA PacketCable keys MUST be F4 - 65537.

8.1.3 Extensions

The following extension MUST be used as specified in the sections below. Any other certificate extensions MAY also be included as non-critical. The encoding tags are [c:critical, n:non-critical; m:mandatory, o:optional] and these are identified in the table for each certificate.

8.1.3.1 subjectKeyIdentifier

The subjectKeyIdentifier extension included in all PacketCable CA certificates as required by RFC 2459 (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 [52]).

8.1.3.2 authorityKeyIdentifier

The authorityKeyIdentifier extension included in all PacketCable certificates as required by RFC 2459 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 [52]).

8.1.3.3 cRLDistributionPoints

TBD

8.1.3.4 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. The end-entity certificates may use the key usage extension as listed in RFC 2459 [52].

8.1.3.5 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.

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

Per [52], the countryName component of an RDN MUST be encoded as a valid two-character PrintableString. That two-character string MUST be a valid ISO country code digraph. All other DirectoryString components of an RDN MUST be encoded as PrintableString where possible (e.g. where the characters of the string are within the set [a-zA-Z0-9'()+,-./:=?]). If a string cannot be encoded as a PrintableString it SHOULD be encoded as a UTF8String (tag [UNIVERSAL 12]). It MAY be encoded as a T61String, but this practice is deprecated in PKIX and may be obsoleted by this standard in future revisions. The string MUST NOT be encoded as a BMPString.

All compliant systems MUST be able to accept any valid encodings of PrintableString, UTFString and T61String, even if they are unable to generate them.

8.2 Certificate Trust Hierarchy

There are two distinct certificate hierarchies used in PacketCable, the MTA Device Certificate Hierarchy and the PacketCable Telephony Certificate Hierarchy.

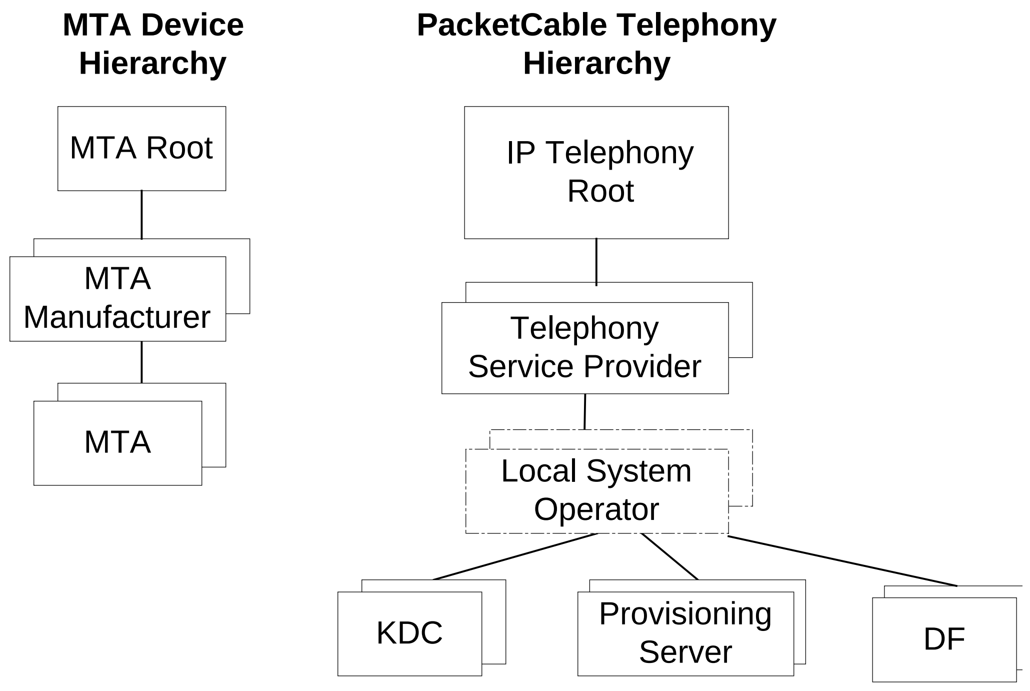

Figure 23. PacketCable Certificate Hierarchy

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 then the certificate serial number, validity period and the value of the signature, exist in the MTA Root certificate that was passed over the wire in comparison to the known MTA Root certificate, the device making the comparison MUST fail the certificate verification. If changes, other then the certificate serial number, validity period and the value of the signature, exist in the IP Telephony Root certificate that was passed over the wire in comparison to the known IP Telephony Root certificate, the device making the comparison MUST fail the certificate verification.

The exact rules for certificate chain validation MUST fully comply with [52], where they are referred to as “Certificate Path Validation”. 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.

The PacketCable validation of validity periods for nesting is not checked and intentionally not enforced, which is compliant with current standards. The validity start date for any end-entity CA MUST be the same as or later then the start date of the issuing CA certificate validity period. The validity end date for end-entities may be before, the same as or after the validity end date for the issuing CA as specified in the PacketCable Certificate tables.

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 table contains the PacketCable specific values for the required fields according to RFC 2459. These PacketCable specific values MUST be followed according to the table below. If a required field is not specifically listed for PacketCable then the guidelines in RFC 2459 MUST be followed. The generic extensions for PacketCable MUST also be included as specified in section 8.1.3

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 18. 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 Provisioning Server. This certificate is not used by the MTAs and thus does not appear in the MTA MIB.
	Signed By	Self-Signed
	Validity Period	30 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), 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 MUST be included in the MTA secure code download as specified by the PacketCable Security Specification and 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 RFC2459 [52].

Table 19. MTA Manufacturer Certificate

MTA Manufacturer Certificate		
	Subject Name Form	C=<country>, O=<CompanyName>, [S=<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 provisioning.
	Signed By	MTA Root Certificate CA
	Validity Period	5 Years, Reissued bi-annually
	Modulus Length	1024, 1536, or 2048
	Extensions	keyUsage[c,m](keyCertSign, cRLSign), 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 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.

An MTA device certificate is permanently installed and not renewable or replaceable. Therefore, it must have a validity period greater than the operational lifetime of the MTA.

Table 20. MTA Device Certificate

MTA Device Certificate		
	Subject Name Form	C=<country>, O=<Company Name>, [S=<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	30 Years
	Modulus Length	1024
	Extensions	keyUsage[n,o](digitalSignature, keyEncipherment) The keyUsage tag is optional. When it is used it SHOULD be marked as critical.

8.2.2.4 MTA Manufacturer Code Verification Certificate

Code Verification Certificate (CVC) specification for embedded MTAs MUST be identical to the DOCSIS 1.1 CVC, specified in [13].

8.2.3 PacketCable Telephony Certificate Hierarchy

The Service Provider Certificate Hierarchy is rooted at a CableLabs issued IP Telephony 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 PacketCable specific values for the required fields according to RFC 2459. These PacketCable specific values MUST be followed according to the table below. If a required field is not specifically listed for PacketCable then the guidelines in RFC 2459 MUST be followed. The generic extensions for PacketCable MUST also be included as specified in section 8.1.3

8.2.3.1 IP Telephony Root Certificate

This root certificate is intended for use only during the MTA provisioning. An MTA receives a configuration file from the provisioning server that includes a digital signature and a certificate chain. The certificate chain includes a Service Provider certificate that is signed by the IP Telephony Root. This certificate MUST be verified as part of a certificate chain containing the IP Telephony Root Certificate, Service Provider Certificate and the Ancillary Device Certificates.

Table 21. IP Telephony Root Certificate

IP Telephony Root Certificate		
	Subject Name Form	C=US, O=CableLabs, OU=PacketCable, CN=PacketCable Root IP Telephony Certificate Authority
	Intended Usage	This certificate is used to sign Telephony Service Provider certificates. It is used by an MTA to verify Provisioning Server certificate chain associated with a signature on a configuration file. 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	30 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), basicConstraints[c,m](cA=true, pathLenConstraint=2)

8.2.3.2 Telephony Service Provider Certificate

This is the certificate held by the telephony service provider, signed by the IP Telephony Root CA. This certificate is the root of all intra-domain trust. During all intra-domain certificate exchanges (with the exception of the MTA provisioning) both parties are already configured with the Telephony Service Provider Certificate and MUST NOT accept a different Telephony Service Provider certificate.

One exception is when the validity period of the Telephony Service Provider certificate is extended and otherwise no changes are made to the contents of the certificate. (As specified in [52], each new certificate will also have a new (and unique to the issuing CA) serial number and a new signature.) In that one case, a new Telephony Service Provider certificate MAY be substituted for the old (possibly expired) certificate. The signature on the new certificate MUST be verified.

This certificate MUST be verified as part of a certificate chain containing the IP Telephony Root Certificate, Service Provider Certificate and the Ancillary Device Certificates.

Table 22. Telephony Service Provider Certificate

Telephony Service Provider Certificate		
	Subject Name Form	C=<country>, O=<Company>, OU=PacketCable, CN=<Company> PacketCable System Operator CA
	Intended Usage	This certificate serves as the root of the intra-domain trust hierarchy. Each network element is configured with this certificate so that all intra-domain communication is limited to this one service provider. This certificate appears as a read-write parameter in the MTA MIB. The provisioning server needs the ability to update this certificate in the MTAs via both SNMP and configuration files. Since in the future each MTA port could be configured with a different telephony service provider, this certificate is associated with an MTA port.
	Signed By	Signed by IP Telephony Root Certificate
	Validity Period	10 years, Re-issued every 5 years
	Modulus Length	2048
	Extensions	keyUsage[c,m](keyCertSign, cRLSign), basicConstraints[c,m](cA=true, pathLenConstraint=1)

8.2.3.3 Local System Certificate

This is the certificate held by the local system. The existence of this certificate is optional, as the Telephony Service Provider CA may be used to directly sign all MTA and network server certificates.

When this certificate exists in the telephony chain, this certificate MUST be verified as part of a certificate chain containing the IP Telephony Root Certificate, Service Provider Certificate and the Ancillary Device Certificates.

Table 23. Local System Certificate

Local System Certificate		
	Subject Name Form	C=<Country>, O=<Company>, OU=PacketCable, OU=<Local System Name>, CN=<Company> PacketCable Local System CA
	Intended Usage	Telephony Service Provider CA may delegate the issuance of certificates to a regional Certification Authority called Local System CA (with the corresponding Local System Certificate). This certificate appears as a read-write parameter in the MTA MIB. The provisioning server needs the ability to update this certificate in the MTAs via both SNMP and configuration files. Since in the future each MTA port could be configured with a different telephony service provider, this certificate is associated with an MTA port. The Local System Certificate that is used to sign a KDC certificate may be different from this MTA MIB variable.
	Signed By	Telephony Service Provider Certificate
	Validity Period	5 years, Re-issued bi-annually
	Modulus Length	1024, 1536, 2048
	Extensions	keyUsage[c,m](keyCertSign, cRLSign), basicConstraints[c,m](cA=true, pathLenConstraint=0)

8.2.4 Operational Ancillary Certificates

All of these are signed by the either the Local System CA or by the Telephony Service Provider CA. Other ancillary certificates may be added to this standard at a later time.

The information contained in the following table contains the PacketCable specific values for the required fields according to RFC 2459. These PacketCable specific values MUST be followed according to the table below. If a required field is not specifically listed for PacketCable then the guidelines in RFC 2459 MUST be followed. The generic extensions for PacketCable MUST also be included as specified in section 8.1.3

8.2.4.1 Key Distribution Center Certificate

This certificate MUST be verified as part of a certificate chain containing the IP Telephony Root Certificate, Service Provider Certificate and the Ancillary Device Certificates.

The PKINIT specification in Appendix B 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.

The encoding of the Kerberos PKINIT subjectAltName is:

```
SEQUENCE {              -- subjectAltName
    [0] SEQUENCE {        -- otherName
      OBJECT IDENTIFIER,   --  1.3.6.1.5.2.2, which is { krb5 2}
      [0] EXPLICIT SEQUENCE {          -- KerberosName
        [0] Realm,            -- Realm: a realm for which this Ticket Granting Service
                              -- issues tickets.
        [1] CertPrincipalName
      }
    }
  }
```

where CertPrincipalName is as defined by [19]:

```
      CertPrincipalName ::= SEQUENCE {
                  name-type[0]     INTEGER,
                  name-string[1]   SEQUENCE OF UTF8String
 -- namestring has two components:
 -- the first element is the string "krbtgt", without the quotation marks
 -- the second element is the name of the local realm.  Note that in the case of
 -- cross-realm operation this realm may not be the same as the realm for which
 -- tickets are issued (inside the KerberosName).  This does not currently apply
 -- to PacketCable because it does not have any PKINIT clients that obtain cross-
 -- realm tickets.
      }
```

Refer to Appendix B for a more detailed specification of the **subjectAltName** syntax.

Table 24. Key Distribution Center Certificate

Key Distribution Center Certificate		
	Subject Name Form	C=<Country>, O=<Company>, OU=PacketCable, OU=[<Local System Name>], OU= 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	Telephony Service Provider Certificate or Local System Certificate
	Validity Period	5 years, reissued annually
	Modulus Length	1024
	Extensions	keyUsage[n,o](digitalSignature), The keyUsage tag is optional. When it is used it SHOULD be marked as critical. subjectAltName[n,m](see PKINIT Spec)

8.2.4.2 Provisioning Server

This certificate MUST be verified as part of a certificate chain containing the IP Telephony Root Certificate, Service Provider Certificate and the Ancillary Device Certificates.[8]

Table 25. Provisioning Server Certificate

Provisioning Server Certificate		
	Subject Name Form	C=<Country>, O=<Company>, OU=PacketCable, OU=[<Local System Name>], OU=Provisioning Servers, CN=<DNS Name>
	Intended Usage	To authenticate the identity of the Provisioning Server to the MTA when it receives a signed configuration file. This certificate is passed to the MTA with the signed configuration file and is therefore not included in the MTA MIB and cannot be updated or queried by the Provisioning Server. At the time that the MTA receives the configuration file it does not know the identities of either the provisioning server or the service provider. Therefore, the MTA will accept any provisioning server identity - as long as the corresponding certificate chain can be verified with the IP Telephony Root Certificate.
	Signed By	Telephony Service Provider Certificate or Local System Certificate
	Validity Period	5 years, reissued annually
	Modulus Length	1024
	Extensions	KeyUsage[n,o](digitalSignature) The keyUsage tag is optional. When it is used it SHOULD be marked as critical.

[8] Currently a PacketCable MTA cannot be relied upon to save a Telephony Service Provider Certificate in non-volatile memory after initial provisioning. Due to this limitation the MTA has to verify the Provisioning Server's certificate chain starting with the IP Telephony Root Certificate each time (where the IP Telephony Root Certificate is installed with the secure code download as specified by the PacketCable Security Specification).

8.2.4.3 Distribution Function (DF)

This certificate MUST be verified as part of a certificate chain containing the IP Telephony Root Certificate, Service Provider 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 a standard dot notation, e.g. 245.120.75.22.

The extendedKeyUsage field (commonly called "EKU") MUST be present and MUST contain only the object identifier iKEIntermediate:

iso.org.dod.internet.security.mechanisms.ipsec.certificate.2 (1.3.6.1.5.5.8.2.2).

Table 26. DF Certificate

DF Certificate		
	Subject Name Form	C=<Country>, O=<Company>, OU=PacketCable, OU=[<Local System Name>], OU=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). For intra-domain Security Associations, these certificates are trusted only if the trust extends to the same Telephony Service Provider Certificate. The inter-domain case is not covered by this version of the specification.
	Signed By	Telephony Service Provider Certificate or Local System Certificate
	Validity Period	5 years, reissued annually
	Modulus Length	1024

	Extensions	KeyUsage[n,o](digitalSignature) The keyUsage tag is optional. When it is used it SHOULD be marked as critical. subjectAltName[n,m](dNSName=<DNSName>) extendedKeyUsage[n,m](iKEIntermediate)

Note that this particular OID is not part of the RFC 2459 certificate profile. It is instead taken from the IPSec certificate profile draft [45].

8.2.4.4 Operator Code Verification Certificate

Code Verification Certificate (CVC) specification for embedded MTAs MUST be identical to the DOCSIS 1.1 CVC, specified in [13].

8.2.5 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 DES

The Data Encryption Standard (DES) is specified in [45]. 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 7.6.2.2.3.1.

9.1.1 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 [48]. 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.

9.1.2 DES-CBC-PAD

This variant of DES is also based on the analysis of DESX presented in [48]. 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 [45], 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 9.1.4), 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.

9.1.3 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.

9.1.4 Block Termination

If block ciphers are supported, a short block (p-bits < 64-bits) MUST be terminated by residual block termination as shown a figure below. Residual block termination (RBT) is executed as follows:

Given a final block having n bits, where n is less than 64, the n bits are padded up to a block of 64 bits by appending (64 – n) bites of arbitrary value to the right of the n-bits. The resulting block is DES encrypted using CFB64 mode, with the next-to-last ciphertext block serving as the initialization vector for the CFB64 operation (see [46] for a description of the CFB64 mode of DES). 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 64-bits, the procedure is the same as for a short final block, with the provided initialization vector serving as the initialization vector for the DES-CFB64 operation. Residual block termination is illustrated in the figure below for both encryption and decryption operations.

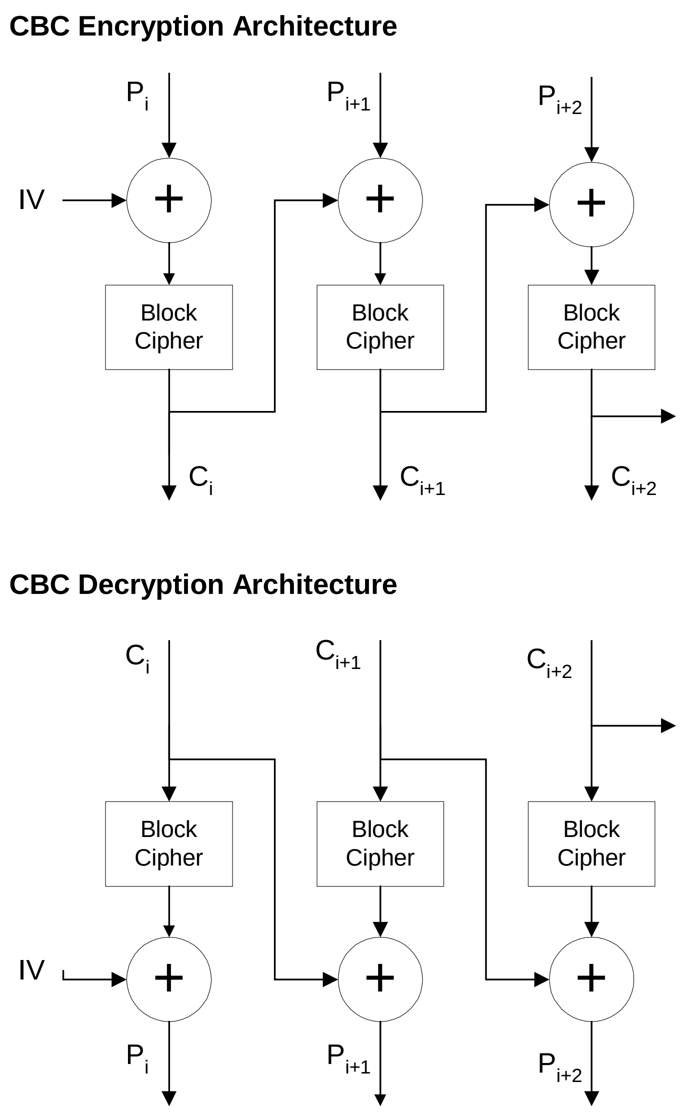

Figure 24. CBC Mode

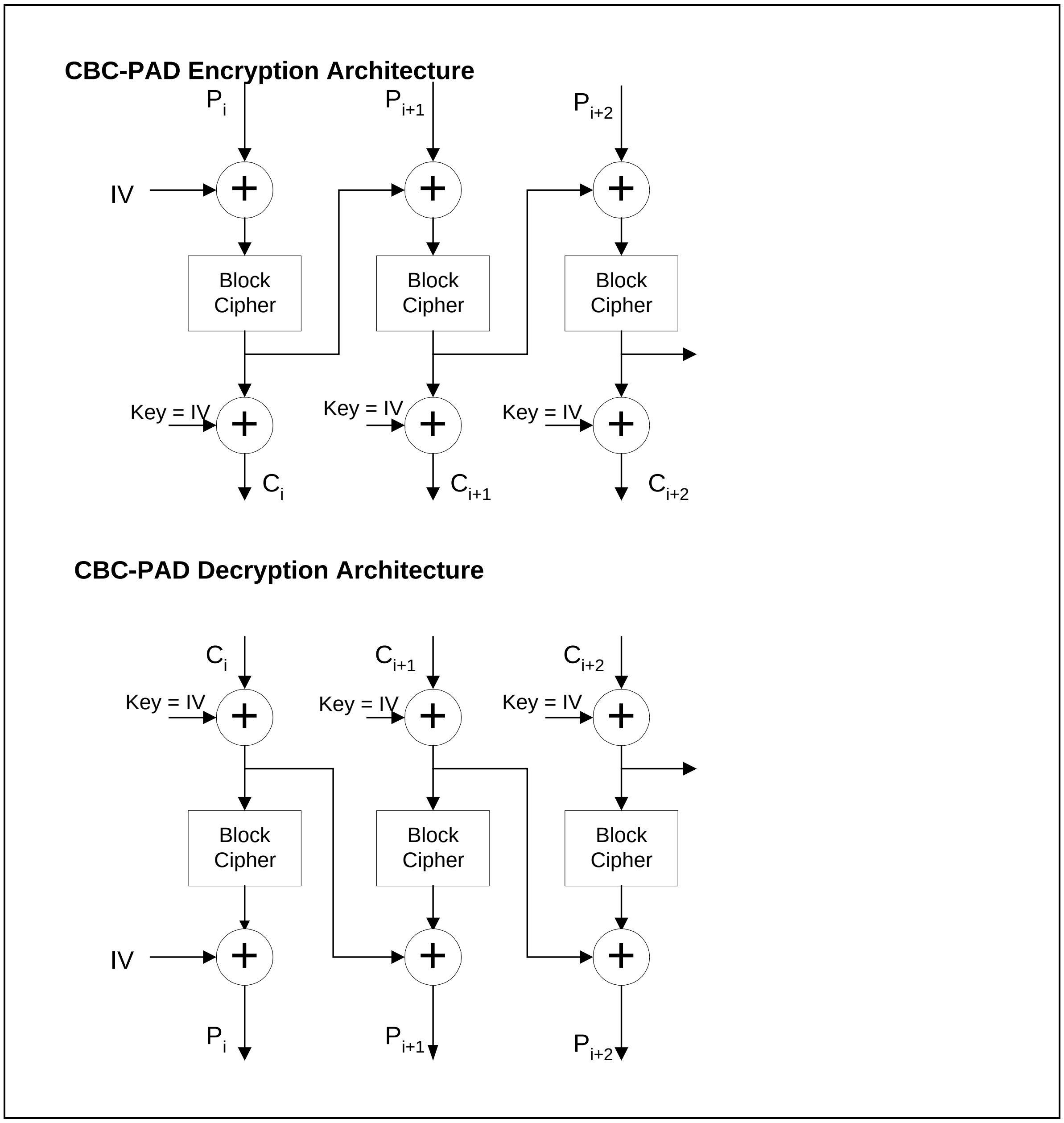

Figure 25. CBC Pad Mode

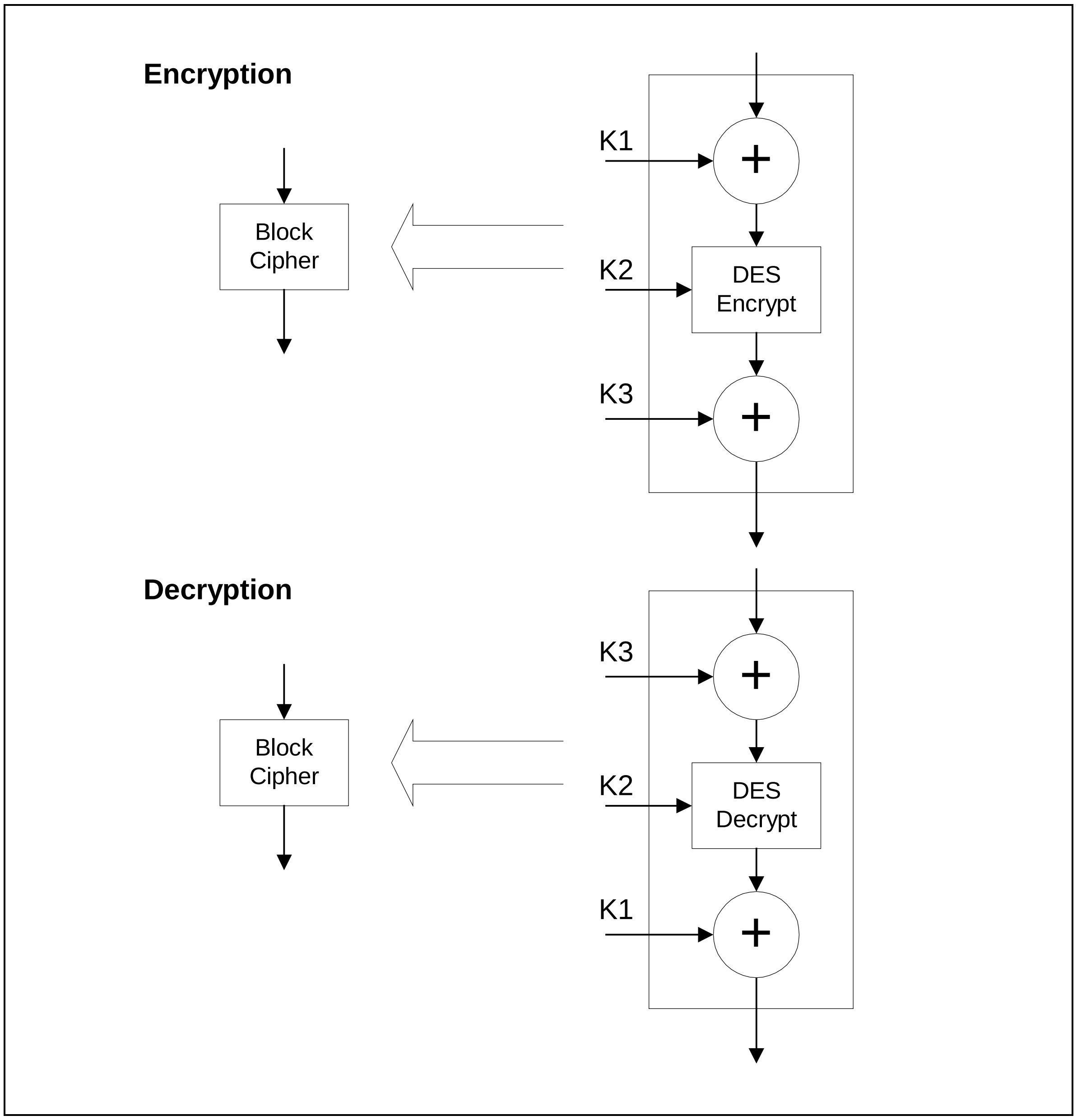

Figure 26. DESX-XEX as Block Cipher

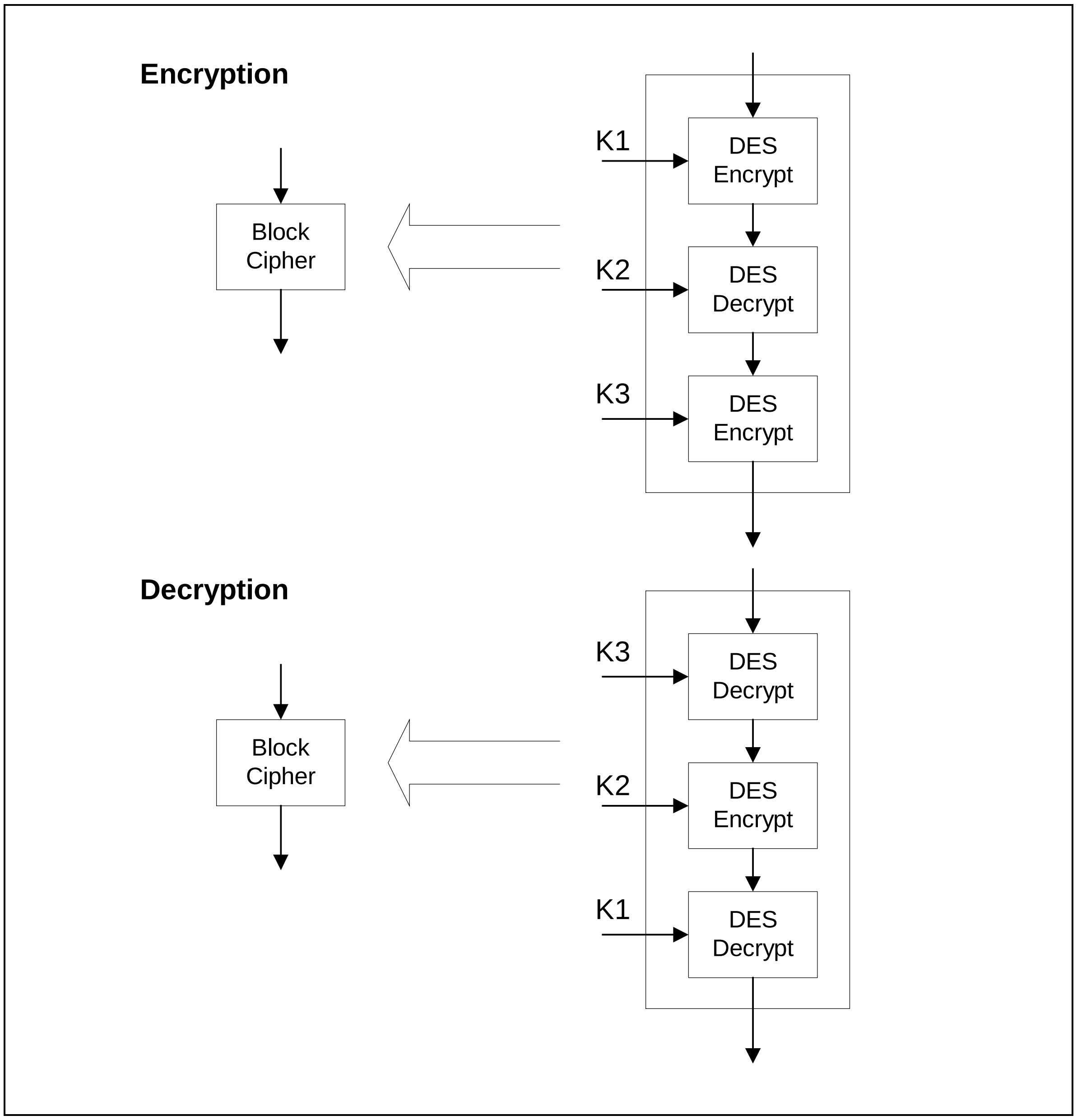

Figure 27. 3DES-EDE as Block Cipher

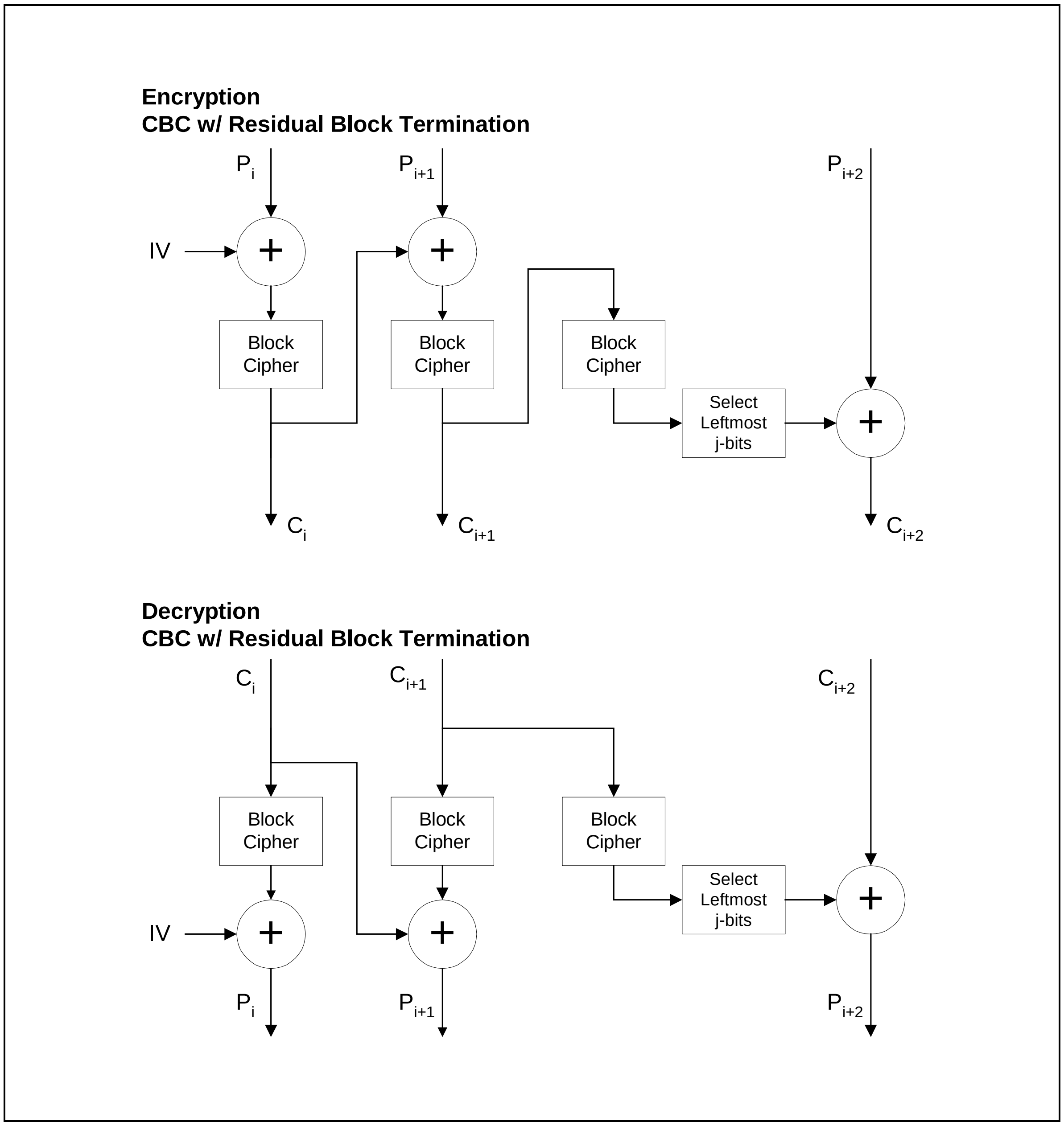

Figure 28. CBC with Residual Block Termination

9.2 RC4

RC4 [1] is a very efficient symmetric cipher. RC4 is used in section 7.6 to encrypt media flows. Key management is described in section 7.6.2.2. The algorithm uses variable length keys. For PacketCable the key length MUST be set to 128 bits. The generation of this 128-bit key from the session key is described in section 9.2.

RC4 is a pseudo-random number generator in output feedback mode. A stream is generated from the key and XORed with the plaintext. There are no integrity protections on the data. RC4 uses a 256-entry substitution box (Sbox), which must be initialized. The entries in the Sbox are represented as bytes S_0, S_1, …, S_{255}. To initialize the Sbox, it is first filled with each entry matching its index. So,

$S_0 = 0$, $S_1 = 1$, … $S_{255} = 255$

Another 256-byte array is filled with the key, repeating as necessary to fill the array, K_0, K_1, … , K_{255}. An index, j, is set to 0. Then, the Sbox is filled as follows:

```
for i = 0 to 255
        $j = (j + S_i + K_i) \bmod 256$
        swap $S_i$ and $S_j$
```

After the loop completes, the Sbox is initialized. The Sbox is dependent on the key, so a new one must be initialized for each key. The Sbox can now be used to generate a pseudo-random stream. i and j are initialized to 0, and a random byte is produced as follows:

```
$i = (i + 1) \bmod 256$
$j = (j + S_i) \bmod 256$
swap $S_i$ and $S_j$
$t = (S_i + S_j) \bmod 256$
random_byte = $S_t$
```

To generate more bytes, the process is repeated using the values for i and j that result from the previous iteration.

9.3 RSA Signature

All public key signatures for PacketCable MUST be generated and verified using the RSA signature algorithm PKCS#1v1.5 [24]. The format for all PacketCable signatures MUST be the RSA Cryptographic Message Syntax [26].

9.4 HMAC-SHA1

The keyed hash employed by the HMAC-Digest Attribute MUST use the HMAC message authentication method [17] with the SHA-1 hash algorithm [23].

HMAC-SHA1 is used to authenticate CMTS-CMS UDP messages for DQoS, as described in section 7.2.1.2.2. The key is 160 bits long. The key management for the HMAC keys is described in section 7.2.1.4.1.

9.5 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 [27]:

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.

9.6 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.

9.6.1 The MMH Function

The Multilinear Modular Hash (MMH) Function described below is a variant of the MMH Function described in [28]. Some of the computations described below use signed arithmetic whereas the computations in [28] 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 [28] 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.

9.6.1.1 MMH[16,s,1]

For the remainder of this section 9.6, 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 9.6.2. The key is denoted by k and the ith word of the key by k_i: $k=k_1,k_2,\ldots,k_s$. Likewise the input message is denoted by m and the ith word of the input message by m_i: $m = m_1, m_2,\ldots, m_s$.

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: $\{-n/2,\ldots,0,\ldots,(n/2)-1\}$. For example, $S_{2^{16}} = \{-2^{15},\ldots,0,\ldots,2^{15}-1\}$ 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 $H_1(k,m) = \; {}_{i=1}^{s} k_i \cdot m_i \; s\bmod 2^{32}$.
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,

$$H(k,m) = \left(\left(\left(\left({}_{i=1}^{s} k_i \cdot m_i \right) s\bmod 2^{32} \right) \bmod p \right) \bmod 2^{16}\right.$$

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 $S_{2^{32}}$.[9] 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

$$a2^{16}+b \equiv a2^{16} + b - a(2^{16}+1) \pmod{(2^{16} + 1)}$$

[9] The entire sum need not be computed before performing the smod 2^{32} operation. The smode 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}.

it follows that $x \equiv b\text{-}a$ (mod p). The quantity $b\text{-}a$ is in the range $\{-2^{15}+1,\ldots,2^{16}-1\}$. Therefore if $b\text{-}a$ is non-negative then x mod $p = b\text{-}a$. If $b\text{-}a$ is negative then x mod $p = b\text{-}a+p$.

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

$a2^{16} + b - 2^{32} \equiv b + a2^{16} - a(2^{16}+1) - 2^{32} + 2^{16}(2^{16} + 1)$ (mod $(2^{16}+1)$)

it follows that $x \equiv b - a + 2^{16}$ (mod p). The quantity $b - a + 2^{16}$ is in the range $\{2^{15} + 1,\ldots,2^{17} - 1\}$. Therefore, if $b - a < p$ then x mod $p = b- a$. If $b - a \geq p$ then x mod $p = b - a- p$.

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.

9.6.1.2 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, $H'(k,m) = H(k^{(1)},m) \circ H(k^{(2)},m)$.

9.6.2 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: $\omega = 16$. 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 ith message sent and received. For the ith 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}

9.6.2.1 MMH-MAC When Using RC-4

When calculating the MMH-MAC for use with RC4, the sequence of key streams is generated by an RC4 key stream as described in section 7.6.2. The $2(s + t - 1)$-byte key for MMH-MAC[16,s,t] are randomly generated as described in section 7.6.2 from a session key for the media flows that is generated by the key agreement protocol give in section 7.6.2.2.

9.6.2.2 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.

9.6.2.3 Odd Payload Sizes

If a message m is not of length s words, but rather of length $v < s$ words, then the input to M is a new message m' given by $m' = m_1,\dots m_v,e_{v+1},\dots e_s$ where $e_{v+1} = \dots = e_s$ is the all zeroes word.

9.7 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 [2] 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

The PacketCable security specification requires that an embedded MTA (MTA-E) and a standalone MTA (MTA-S) maintain persistent IPSec encryption and authentication keys and Kerberos session keys. An MTA MUST also 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 140-1, Security Requirements for Cryptographic Modules, standard. An MTA-E or MTA-S SHOULD, at a minimum meet FIPS PUBS 140-1 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 network.

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 140-1 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 140-1 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.

Two of the (many) requirements of FIPS PUB 140-1 Security Level 3 recommends “data ports for critical security parameters be physically separated from other data ports” and, entry/exit of keys in encrypted form or direct entry/exit with split knowledge procedures”. As also mentioned in the previous section, the PacketCable security specification is not requiring compliance with any of the FIPS PUB 140-1 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 UPGRADE

The scope of PacketCable includes only Embedded MTAs. Therefore PacketCable MTAs MUST be embedded with the DOCSIS 1.1 cable modem that MUST implement BPI+. PacketCable Embedded MTAs MUST have their software upgraded according to the DOCSIS 1.1 requirements. The cable modem will verify the code file using DOCSIS parameters that include the DOCSIS root key and the Code Verification Certificate (CVC). Requirements for DOCSIS 1.1 software upgrade are specified in references [11] and [14].

The future implementation of secure software upgrades for Standalone MTAs is expected to utilize a similar method for secure software upgrades. The details of these requirements will be defined in a subsequent version of this specification.

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.

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 a provisioned parameter that can be use in one the following ways:

In the case 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. PKINIT Specification

The PKINIT specification is currently still an IETF draft. This document complies only with the version of the PKINIT draft that is listed in this section. The PacketCable security team will continue to track progress of the PKINIT draft through the IETF.

```
INTERNET-DRAFT                                           Brian Tung
draft-ietf-cat-kerberos-pk-init-13.txt                         Clifford
Neuman
Updates: RFC 1510
USC/ISIexpires January 15, 2001                                Matthew Hur
                                                         CyberSafe
Corporation
                                                         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-13.txt, and expires January 15, 2001. Please send comments to the authors.

1. Abstract

This document defines extensions (PKINIT) to the Kerberos protocol specification (RFC 1510 [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 digital signature keys as the primary, required mechanism. It also allows for the use of RSA keys and/or (static) Diffie-Hellman certificates. Note in particular 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 secret key, a public key, or both.

The PKINIT specification may also be used as a building block for other specifications. PKCROSS [3] utilizes PKINIT for establishing the inter-realm key and associated inter-realm policy to be applied in issuing cross realm service tickets. As specified in [4], anonymous Kerberos tickets can be issued by applying a NULL signature in combination with Diffie-Hellman in the PKINIT exchange. Additionally, the PKINIT specification may be used for direct peer to peer authentication without contacting a central KDC. This application of PKINIT is described in PKTAPP [5] and 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 [8] 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 [9]).

3. Proposed Extensions

This section describes extensions to RFC 1510 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 1510 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).

 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 RFC1510.
 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 [14] (part of LDAPv3 [18]).

To ensure that this encoding is unique, we add the following rule to those specified by RFC 2253:

The order in which the attributes appear in the RFC 2253 encoding must be the reverse of the order in the ASN.1 encoding of the X.500 name that appears in the public key certificate. The order of the relative distinguished names (RDNs), as well as the order of the AttributeTypeAndValues within each RDN, will be reversed. (This is despite the fact that an RDN is defined as a SET of AttributeTypeAndValues, where an order is normally not important.)

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 shall be set to KRB_NT-X500-PRINCIPAL. This new name type is defined in RFC 1510 as:

```
KRB_NT_X500_PRINCIPAL 6
```

The name-string shall be set as follows:

```
RFC2253Encode(DistinguishedName)
```

as described above. When this name type is used, the principal's realm shall be set to the certificate authority's distinguished name using the X500-RFC2253 realm name format described earlier in this section

RFC 1510 specifies the ASN.1 structure for PrincipalName as follows:

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

For the purposes of encoding an X.500 name as a Kerberos name for use in Kerberos structures, the name-string shall be encoded as a single GeneralString. The name-type should be KRB_NT_X500_PRINCIPAL, as noted above. All Kerberos names must conform to validity requirements as given in RFC 1510. Note that name mapping may be required or optional, based on policy.

We also define the following similar ASN.1 structure:

```
CertPrincipalName ::= SEQUENCE {
name-type[0] INTEGER,
name-string[1] SEQUENCE OF UTF8String
}
```

When a Kerberos PrincipalName is to be placed within an X.509 data structure, the CertPrincipalName structure is to be used, with the name-string encoded as a single UTF8String. The name-type should be as identified in the original PrincipalName structure. The mapping between the GeneralString and UTF8String formats can be found in [19].

The following rules relate to the matching of PrincipalNames (or corresponding CertPrincipalNames) with regard to the PKI name constraints for CAs as laid out in RFC 2459 [15]. 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 user plus instance plus realm name (as specified in RFC 1510), 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 contains an equal sign, it is treated as an X.500 name. In order to match, every component in the constraint MUST be in the principal name, and have the same value. For example, 'C=US' matches 'C=US/O=ISI' but not 'C=UK'.

c. 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.

d. 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 shall be produced from the agreed bit string as follows:

* Truncate the bit string to the appropriate length.
* Rectify parity in each byte (if necessary) to obtain the key.

For instance, in the case of a DES key, we take the first eight bytes of the bit stream, and then adjust the least significant bit of each byte to ensure that each byte has odd parity.

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 [11] and in [15] shall be used with PKINIT:

```
id-dsa-with-sha1 (DSA with SHA1)
md5WithRSAEncryption (RSA with MD5)
sha-1WithRSAEncryption (RSA 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 [15].

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 [16]. Currently, only PKCS#1 v1.5 is specified by CMS [11], 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 [11].

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 1510, 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 [11];
-- identified by the
-- SignedData OID: {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 [11];
-- 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
}
```

Usage of SignedData (encapsulated within signedAuthPack):

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 shall 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.

```
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 RFC1510
ctime [1] KerberosTime,
-- for replay prevention as in RFC1510
nonce [2] INTEGER,
pachecksum [3] Checksum
-- Checksum over KDC-REQ-BODY
-- Defined by Kerberos spec
}

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 [10]

AlgorithmIdentifier ::= SEQUENCE {
algorithm OBJECT IDENTIFIER,
-- for dhKeyAgreement, this is
-- { iso (1) member-body (2) US (840)
-- rsadsi (113459) pkcs (1) 3 1 }
-- from PKCS #3 [20]
parameters ANY DEFINED by algorithm OPTIONAL
-- for dhKeyAgreement, this is
-- DHParameter
} -- as specified by the X.509 recommendation [10]

DHParameter ::= SEQUENCE {
prime INTEGER,
-- p
base INTEGER,
-- g
privateValueLength INTEGER OPTIONAL
-- l
} -- as defined in PKCS #3 [20]
```

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 1510 revisions as a SEQUENCE of TypedData:

```
TypedData ::= SEQUENCE {
data-type [0] INTEGER,
data-value [1] OCTET STRING,
} -- per Kerberos RFC 1510 revisions
```

```
where:
data-type = TD-PKINIT-CMS-CERTIFICATES = 101
data-value = CertificateSet // as specified by CMS [11]
```

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. This may be based on a certification hierarchy, or it may be simply a list of recognized certifiers in a system like PGP.

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 that 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.

Finally, if the certificate chain is verified, but the KDC's name or realm as given in the PKAuthenticator does not match the KDC's actual principal name, then the KDC returns an error of type KDC_ERR_KDC_NAME_MISMATCH. The accompanying e-data field is again a SEQUENCE of one TypedData (with type TD-KRB-PRINCIPAL=102 or TD-KRB-REALM=103 as appropriate) whose data-value is an OCTET

```
 STRING whose data-value is the DER encoding of a PrincipalName or
 Realm as defined in RFC 1510 revisions.

 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 (4).

 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. 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 1510.

 Assuming no errors, the KDC replies as per RFC 1510, 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
 mapping as necessary (e.g., as per RFC 2253 for X.500
 names). In this case the realm in the ticket shall be the
 name of the certifier that issued the user's certificate.

 Note that a principal name may be carried in the subject alt name
 field of a certificate. This name may be mapped to a principal
 record in a security database based on local policy, for example
 the subject alt name 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 [17], 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
 canonicalized. 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 the client's CA (if in the middle of a CA key roll-over, use the KDC cert issued under same CA key as user cert used to verify request).
3. 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 [11] and used only with
-- Diffie-Hellman key exchange (if the
-- client public value was present in the
-- request).
-- Again, SignedData OID is {pkcs7 2}
-- This choice MUST be supported
-- by compliant implementations.
encKeyPack [1] ContentInfo
-- Defined in CMS [11]
-- EnvelopedData OID is {pkcs7 3}
-- The temporary key is encrypted
-- using the client public key
-- key.
-- SignedReplyKeyPack, encrypted
-- with the temporary key, is also
-- included.
}
```

Usage of SignedData (encapsulated within dhSignedData):

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.

3. If the reply key is DES, N=64 bits, where some of the bits are replaced with parity bits, according to FIPS PUB 74.

4. 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.

5. The encapContentInfo field must contain the KdcDHKeyInfo as defined below.

a. The eContentType field shall 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).

6. 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).

7. The signerInfos field is a SET that must contain at least one member, since it contains the actual signature.

```
KdcDHKeyInfo ::= SEQUENCE {
-- used only when utilizing Diffie-Hellman
nonce [0] INTEGER,
-- binds response to the request
subjectPublicKey [2] BIT STRING
-- Equals public exponent (g^a mod p)
-- INTEGER encoded as payload of
-- BIT STRING
}
```

Usage of EnvelopedData:

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 that must contain exactly one member of the KeyTransRecipientInfo type for encryption with an RSA 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 shall 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 shall 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,
-- 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
}

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.

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 shall contain the principal name of the KDC
(defined in section 8.2 of RFC 1510) 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 shall be a KerberosName as defined in RFC 1510, but with
the PrincipalName replaced by CertPrincipalName as mentioned in
```

```
 Section 3.1:

 KerberosName ::= SEQUENCE {
 realm [0] Realm,
 principalName [1] CertPrincipalName -- defined above
 }

 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.

 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 1510.

3.2.3. 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 DSA for signatures
 * SHA1 digest also for the Checksum in the PKAuthenticator
 * 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 introduces a new trust model, where KDCs do not (necessarily) certify the identity of those for whom they issue tickets. PKINIT does allow KDCs to act as their own CAs, in the limited capacity of self-signing their certificates, but one of the additional benefits is to align Kerberos authentication with a global public key infrastructure. Anyone using PKINIT in this way must be aware of how the certification infrastructure they are linking to works.

Secondly, PKINIT also introduces the possibility of interactions between different cryptosystems, which may be of widely varying strengths. 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, is inappropriate; it adds a weak link to a strong one at extra cost. Implementors and administrators should take care to avoid such wasteful and deceptive interactions.

Lastly, PKINIT calls for randomly generated keys for conventional cryptosystems. Many such systems contain systematically "weak" keys. PKINIT implementations MUST avoid use of these keys, either by discarding those keys when they are generated, or by fixing them in some way (e.g., by XORing them with a given mask). These precautions vary from system to system; it is not our intention to give an explicit recipe for them here.

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] B. Tung, T. Ryutov, C. Neuman, G. Tsudik, B. Sommerfeld, A. Medvinsky, M. Hur. Public Key Cryptography for Cross-Realm Authentication in Kerberos. draft-ietf-cat-kerberos-pk-cross-04.txt

[4] A. Medvinsky, J. Cargille, M. Hur. Anonymous Credentials in Kerberos. draft-ietf-cat-kerberos-anoncred-00.txt

[5] Ari Medvinsky, M. Hur, Alexander Medvinsky, B. Clifford Neuman. Public Key Utilizing Tickets for Application Servers (PKTAPP). draft-ietf-cat-pktapp-02.txt

[6] M. Sirbu, J. Chuang. Distributed Authentication in Kerberos Using Public Key Cryptography. Symposium On Network and Distributed System Security, 1997.

[7] B. Cox, J.D. Tygar, M. Sirbu. NetBill Security and Transaction Protocol. In Proceedings of the USENIX Workshop on Electronic Commerce, July 1995.

[8] T. Dierks, C. Allen. The TLS Protocol, Version 1.0 Request for Comments 2246, January 1999.

[9] B.C. Neuman, Proxy-Based Authorization and Accounting for Distributed Systems. In Proceedings of the 13th International Conference on Distributed Computing Systems, May 1993.

[10] ITU-T (formerly CCITT) Information technology – Open Systems Interconnection – The Directory: Authentication Framework Recommendation X.509 ISO/IEC 9594-8

[11] R. Housley. Cryptographic Message Syntax. draft-ietf-smime-cms-13.txt, April 1999, approved for publication as RFC.

[12] PKCS #7: Cryptographic Message Syntax Standard, An RSA Laboratories Technical Note Version 1.5 Revised November 1, 1993

[13] 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.

[14] M. Wahl, S. Kille, T. Howes. Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names. Request for Comments 2253.

[15] R. Housley, W. Ford, W. Polk, D. Solo. Internet X.509 Public Key Infrastructure, Certificate and CRL Profile, January 1999. Request for Comments 2459.

[16] B. Kaliski, J. Staddon. PKCS #1: RSA Cryptography Specifications, October 1998. Request for Comments 2437.

[17] S. Dusse, P. Hoffman, B. Ramsdell, J. Weinstein. S/MIME Version 2 Certificate Handling, March 1998. Request for Comments 2312.

[18] M. Wahl, T. Howes, S. Kille. Lightweight Directory Access Protocol (v3), December 1997. Request for Comments 2251.

[19] 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

[20] 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 January 15, 2001.

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
CyberSafe Corporation
1605 NW Sammamish Road
Issaquah WA 98027-5378
Phone: +1 425 391 6000
E-mail: matt.hur@cybersafe.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
170 W. Tasman Dr.
San Jose, CA 95134
E-mail: jtrostle@cisco.com

Appendix C. PKCROSS Specification

The PKCROSS specification is currently still an IETF draft. This document complies only with the version of the PKCROSS draft that is listed in this section. The PacketCable security team will continue to track progress of the PKCROSS draft through the IETF.

```
INTERNET-DRAFT                                          Matthew Hur
draft-ietf-kerberos-pk-cross-07.txt                  Cisco Systems
Updates: RFC 1510                                      Brian Tung
expires May 15, 2001                               Tatyana Ryutov
                                                  Clifford Neuman
                                                              ISI
                                                   Ari Medvinsky
                                                         Keen.com
                                                      Gene Tsudik
                                                        UC Irvine
                                                  Bill Sommerfeld
                                                 Sun Microsystems


    Public Key Cryptography for Cross-Realm 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.''

    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-kerberos-pk-cross-07.txt, and expires May 15, 2001.
    Please send comments to the authors.


1.  Abstract

    This document defines extensions to the Kerberos protocol
    specification [1] to provide a method for using public key
    cryptography to enable cross-realm authentication.  The methods
    defined here specify the way in which message exchanges are to be
    used to transport cross-realm secret keys protected by encryption
    under public keys certified as belonging to KDCs.
```

2. Introduction

Symmetric and asymmetric key systems may co-exist within hybrid architectures in order to leverage the advantages and mitigate issues within the respective systems. An example of a hybrid solution that may employ both symmetric and asymmetric technologies is Kerberos ciphersuites in TLS [KERBTLS] which utilizes the Kerberos protocol [KERB] [KERB94] in conjunction with TLS [TLS] which has commonly been thought of as a public key protocol.

The Kerberos can leverage the advantages provided by public key cryptography. PKINIT [PKINIT] describes the use of public key cryptography in the initial authentication exchange in Kerberos. PKTAPP [PKTAPP] describes how an application service can essentially issue a kerberos ticket to itself after utilizing public key cryptography for authentication. This specification describes the use of public key cryptography in cross-realm authentication.

Without the use of public key cryptography, administrators must maintain separate keys for every realm that wishes to exchange authentication information with another realm (which implies n(n-1) keys), or they must utilize a hierarchical arrangement of realms, which may increase network traffic and complicate the trust model by requiring evaluation of transited realms.

Even with the multi-hop cross-realm authentication, there must be some way to locate the path by which separate realms are to be transited. The current method, which makes use of the DNS-like realm names typical to Kerberos, requires trust of the intermediate KDCs.

PKCROSS utilizes a public key infrastructure (PKI) [X509] to simplify the administrative burden of maintaining cross-realm keys. Such usage leverages a PKI for a non-centrally-administratable environment (namely, inter-realm). Thus, a shared key for cross-realm authentication can be established for a set period of time, and a remote realm is able to issue policy information that is returned to itself when a client requests cross-realm authentication. Such policy information may be in the form of restrictions [NEUMAN]. Furthermore, these methods are transparent to the client; therefore, only the KDCs need to be modified to use them. In this way, we take advantage of the distributed trust management capabilities of public key cryptography while maintaining the advantages of localized trust management provided by Kerberos.

Although this specification utilizes the protocol specified in the PKINIT specification, it is not necessary to implement client changes in order to make use of the changes in this document.

3. Objectives

The objectives of this specification are as follows:

1. Simplify the administration required to establish Kerberos cross-realm keys.

2. Avoid modification of clients and application servers.

3. Allow remote KDC to control its policy on cross-realm keys shared between KDCs, and on cross-realm tickets presented by clients.

4. Remove any need for KDCs to maintain state about keys shared with other KDCs.

```
       5.  Leverage the work done for PKINIT to provide the public key
           protocol for establishing symmetric cross realm keys.


   4.  Definitions

       The following notation is used throughout this specification:
       KDC_l ........... local KDC
       KDC_r ........... remote KDC
       XTKT_(l,r) ...... PKCROSS ticket that the remote KDC issues to the
                         local KDC
       TGT_(c,r) ....... cross-realm TGT that the local KDC issues to the
                         client for presentation to the remote KDC

       This specification defines the following new types to be added to
       the Kerberos specification:
         PKCROSS kdc-options field in the AS_REQ is bit 9
         TE-TYPE-PKCROSS-KDC       2
         TE-TYPE-PKCROSS-CLIENT    3

       This specification defines the following ASN.1 type for conveying
       policy information:
       CrossRealmTktData ::= SEQUENCE OF TypedData

       This specification defines the following types for policy
       information conveyed in CrossRealmTktData:
         PLC_LIFETIME              1
         PLC_SET_TKT_FLAGS         2
         PLC_NOSET_TKT_FLAGS       3

       TicketExtensions are defined per the Kerberos specification
       [KERB-REV]:
       TicketExtensions ::= SEQUENCE OF TypedData
         Where
           TypedData ::=   SEQUENCE {
               data-type[0]   INTEGER,
               data-value[1]  OCTET STRING OPTIONAL
           }


   5.  Protocol Specification

       We assume that the client has already obtained a TGT.  To perform
       cross-realm authentication, the client does exactly what it does
       with ordinary (i.e. non-public-key-enabled) Kerberos; the only
       changes are in the KDC; although the ticket which the client
       forwards to the remote realm may be changed.  This is acceptable
       since the client treats the ticket as opaque.


   5.1.  Overview of Protocol

       The basic operation of the PKCROSS protocol is as follows:

           1.  The client submits a request to the local KDC for
               credentials for the remote realm.  This is just a typical
               cross realm request that may occur with or without PKCROSS.

           2.  The local KDC submits a PKINIT request to the remote KDC to
               obtain a "special" PKCROSS ticket.  This is a standard
               PKINIT request, except that PKCROSS flag (bit 9) is set in
               the kdc-options field in the AS_REQ.

           3.  The remote KDC responds as per PKINIT, except that
               the ticket contains a TicketExtension, which contains
               policy information such as lifetime of cross realm tickets
               issued by KDC_l to a client.  The local KDC must reflect
```

```
        this policy information in the credentials it forwards to
        the client.  Call this ticket XTKT_(l,r) to indicate that
        this ticket is used to authenticate the local KDC to the
        remote KDC.

    4.  The local KDC passes a ticket, TGT_(c,r) (the cross realm
        TGT between the client and remote KDC), to the client.
        This ticket contains in its TicketExtension field the
        ticket, XTKT_(l,r), which contains the cross-realm key.
        The TGT_(c,r) ticket is encrypted using the key sealed in
        XTKT_(l,r).  (The TicketExtension field is not encrypted.)
        The local KDC may optionally include another TicketExtension
        type that indicates the hostname and/or IP address for the
        remote KDC.

    5.  The client submits the request directly to the remote
        KDC, as before.

    6.  The remote KDC extracts XTKT_(l,r) from the TicketExtension
        in order to decrypt the encrypted part of TGT_(c,r).

--------------------------------------------------------------------

Client                Local KDC (KDC_l)            Remote KDC (KDC_r)
------                -----------------            ------------------
Normal Kerberos
request for
cross-realm
ticket for KDC_r
---------------------->

                      PKINIT request for
                      XTKT(l,r) - PKCROSS flag
                      set in the AS-REQ
                      * -------------------------->

                                                   PKINIT reply with
                                                   XTKT_(l,r) and
                                                   policy info in
                                                   ticket extension
                                 <-------------------------- *

                      Normal Kerberos reply
                      with TGT_(c,r) and
                      XTKT(l,r) in ticket
                      extension
     <----------------------------------

Normal Kerberos
cross-realm TGS-REQ
for remote
application
service with
TGT_(c,r) and
XTKT(l,r) in ticket
extension
------------------------------------------------->

                                                   Normal Kerberos
                                                   cross-realm
                                                   TGS-REP
    <---------------------------------------------------------------

* Note that the KDC to KDC messages occur only periodically, since
  the local KDC caches the XTKT_(l,r).
--------------------------------------------------------------------
```

Sections 5.2 through 5.4 describe in detail steps 2 through 4 above. Section 5.6 describes the conditions under which steps 2 and 3 may be skipped.

Note that the mechanism presented above requires infrequent KDC to KDC communication (as dictated by policy - this is discussed later). Without such an exchange, there are the following issues:
1) KDC_l would have to issue a ticket with the expectation that KDC_r will accept it.
2) In the message that the client sends to KDC_r, KDC_l would have to authenticate KDC_r with credentials that KDC_r trusts.
3) There is no way for KDC_r to convey policy information to KDC_l.
4) If, based on local policy, KDC_r does not accept a ticket from KDC_l, then the client gets stuck in the middle. To address such an issue would require modifications to standard client processing behavior.

Therefore, the infrequent use of KDC to KDC communication assures that inter-realm KDC keys may be established in accordance with local policies and that clients may continue to operate without modification.

5.2. Local KDC's Request to Remote KDC

When the local KDC receives a request for cross-realm authentication, it first checks its ticket cache to see if it has a valid PKCROSS ticket, XTKT_(l,r). If it has a valid XTKT_(l,r), then it does not need to send a request to the remote KDC (see section 5.5).

If the local KDC does not have a valid XTKT_(l,r), it sends a request to the remote KDC in order to establish a cross realm key and obtain the XTKT_(l,r). This request is in fact a PKINIT request as described in the PKINIT specification; i.e., it consists of an AS-REQ with a PA-PK-AS-REQ included as a preauthentication field. Note, that the AS-REQ MUST have the PKCROSS flag (bit 9) set in the kdc_options field of the AS-REQ. Otherwise, this exchange exactly follows the description given in the PKINIT specification.

5.3. Remote KDC's Response to Local KDC

When the remote KDC receives the PKINIT/PKCROSS request from the local KDC, it sends back a PKINIT response as described in the PKINIT specification with the following exception: the encrypted part of the Kerberos ticket is not encrypted with the krbtgt key; instead, it is encrypted with the ticket granting server's PKCROSS key. This key, rather than the krbtgt key, is used because it encrypts a ticket used for verifying a cross realm request rather than for issuing an application service ticket. Note that, as a matter of policy, the session key for the XTKT_(l,r) MAY be of greater strength than that of a session key for a normal PKINIT reply, since the XTKT_(l,r) SHOULD be much longer lived than a normal application service ticket.

In addition, the remote KDC SHOULD include policy information in the XTKT_(l,r). This policy information would then be reflected in the cross-realm TGT, TGT_(c,r). Otherwise, the policy for TGT_(c,r) would be dictated by KDC_l rather than by KDC_r. The local KDC MAY enforce a more restrictive local policy when creating a cross-realm ticket, TGT_(c,r). For example, KDC_r may dictate a lifetime policy of eight hours, but KDC_l may create TKT_(c,r) with a lifetime of four hours, as dictated by local policy. Also, the remote KDC MAY include other information about itself along with the PKCROSS ticket. These items are further discussed in section 6 below.

5.4. Local KDC's Response to Client

Upon receipt of the PKINIT/CROSS response from the remote KDC, the local KDC formulates a response to the client. This reply is constructed exactly as in the Kerberos specification, except for the following:

```
A) The local KDC places XTKT_(l,r) in the TicketExtension field of
   the client's cross-realm, ticket, TGT_(c,r), for the remote
   realm.
   Where
      data-type   equals 3 for TE-TYPE-PKCROSS-CLIENT
      data-value  is ASN.1 encoding of XTKT_(l,r)

B) The local KDC adds the name of its CA to the transited field of
   TGT_(c,r).
```

5.5 Remote KDC's Processing of Client Request

When the remote KDC, KDC_r, receives a cross-realm ticket, TGT_(c,r), and it detects that the ticket contains a ticket extension of type TE-TYPE-PKCROSS-CLIENT, KDC_r must first decrypt the ticket, XTKT_(l,r) that is encoded in the ticket extension. KDC_r uses its PKCROSS key in order to decrypt XTKT_(l,r). KDC_r then uses the key obtained from XTKT_(l,r) in order to decrypt the cross-realm ticket, TGT_(c,r).

KDC_r MUST verify that the cross-realm ticket, TGT_(c,r) is in compliance with any policy information contained in XTKT_(l,r) (see section 6). If the TGT_(c,r) is not in compliance with policy, then the KDC_r responds to the client with a KRB-ERROR message of type KDC_ERR_POLICY.

5.6. Short-Circuiting the KDC-to-KDC Exchange

As we described earlier, the KDC to KDC exchange is required only for establishing a symmetric, inter-realm key. Once this key is established (via the PKINIT exchange), no KDC to KDC communication is required until that key needs to be renewed. This section describes the circumstances under which the KDC to KDC exchange described in Sections 5.2 and 5.3 may be skipped.

The local KDC has a known lifetime for TGT_(c,r). This lifetime may be determined by policy information included in XTKT_(l,r), and/or it may be determined by local KDC policy. If the local KDC already has a ticket XTKT(l,r), and the start time plus the lifetime for TGT_(c,r) does not exceed the expiration time for XTGT_(l,r), then the local KDC may skip the exchange with the remote KDC, and issue a cross-realm ticket to the client as described in Section 5.4.

Since the remote KDC may change its PKCROSS key (referred to in Section 5.2) while there are PKCROSS tickets still active, it SHOULD cache the old PKCROSS keys until the last issued PKCROSS ticket expires. Otherwise, the remote KDC will respond to a client with a KRB-ERROR message of type KDC_ERR_TGT_REVOKED.

6. Extensions for the PKCROSS Ticket

As stated in section 5.3, the remote KDC SHOULD include policy information in XTKT_(l,r). This policy information is contained in a TicketExtension, as defined by the Kerberos specification, and the authorization data of the ticket will contain an authorization record of type AD-IN-Ticket-Extensions. The TicketExtension defined for use by PKCROSS is TE-TYPE-PKCROSS-KDC.

```
  Where
    data-type   equals 2 for TE-TYPE-PKCROSS-KDC
    data-value  is ASN.1 encoding of CrossRealmTktData

  CrossRealmTktData ::= SEQUENCE OF TypedData
```

CrossRealmTktData types and the corresponding data are interpreted as follows:

type	value	interpretation	ASN.1 data encoding
PLC_LIFETIME	1	lifetime (in seconds) for TGT_(c,r) - cross-realm tickets issued for clients by TGT_l	INTEGER
PLC_SET_TKT_FLAGS	2	TicketFlags that must be set - format defined by Kerberos specification	BITSTRING
PLC_NOSET_TKT_FLAGS	3	TicketFlags that must not be set - format defined by Kerberos specification	BITSTRING

Further types may be added to this table.

----

7. Usage of Certificates

In the cases of PKINIT and PKCROSS, the trust in a certification authority is equivalent to Kerberos cross realm trust. For this reason, an implementation MAY choose to use the same KDC certificate when the KDC is acting in any of the following three roles:

1) KDC is authenticating clients via PKINIT
2) KDC is authenticating another KDC for PKCROSS
3) KDC is the client in a PKCROSS exchange with another KDC

Note that per PKINIT, the KDC X.509 certificate (the server in a PKINIT exchange) MUST contain the principal name of the KDC in the subjectAltName field.

8. Transport Issues

Because the messages between the KDCs involve PKINIT exchanges, and PKINIT recommends TCP as a transport mechanism (due to the length of the messages and the likelihood that they will fragment), the same recommendation for TCP applies to PKCROSS as well.

9. Security Considerations

Since PKCROSS utilizes PKINIT, it is subject to the same security considerations as PKINIT. Administrators should assure adherence to security policy - for example, this affects the PKCROSS policies for cross realm key lifetime and for policy propagation from the PKCROSS ticket, issued from a remote KDC to a local KDC, to cross realm tickets that are issued by a local KDC to a client.

10. Bibliography

[KERBTLS] A. Medvinsky and M. Hur, "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)", RFC 2712, October 1999.

[KERB] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[TLS] T. Dierks and C. Allen, "The TLS Protocol, Version 1.0", RFC 2246, January 1999.

[PKINIT] B. Tung, C. Neuman, M. Hur, A. Medvinsky, S. Medvinsky, J. Wray, J. Trostle. Public Key Cryptography for Initial Authentication in Kerberos. draft-ietf-cat-kerberos-pk-init-12.txt

[PKTAPP] A. Medvinsky, M. Hur, S. Medvinsky, C. Neuman. Public Key Utilizing Tickets for Application Servers (PKTAPP). draft-ietf-cat-kerberos-pk-tapp-03.txt

[X509] ITU-T (formerly CCITT) Information technology - Open Systems Interconnection - The Directory: Authentication Framework Recommendation X.509 ISO/IEC 9594-8

[NEUMAN] B.C. Neuman, "Proxy-Based Authorization and Accounting for Distributed Systems". Proceedings of the 13th International Conference on Distributed Computing Systems, May 1993

[KERB94] B.C. Neuman, Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994.

[KERB-REV] C.Neuman, J. Kohl, T. Ts'o. The Kerberos Network Authentication Service (V5). draft-ietf-cat-kerberos-revisions-07.txt

11. Authors' Addresses

Matthew Hur
Cisco Systems
500 108th Ave. NE, Suite 500
Bellevue, WA 98004
Phone:
EMail: mhur@cisco.com

Brian Tung
Tatyana Ryutov
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, tryutov, bcn}@isi.edu

Ari Medvinsky
Keen.com
2480 Sand Hill Road, Suite 200
Menlo Park, CA 94025
Phone +1 650 289 3134
E-mail: ari@keen.com

```
Gene Tsudik
ICS Dept, 458 CS Building
Irvine CA 92697-3425
Phone: +1 310 448 9329
E-Mail: gts@ics.uci.edu

Bill Sommerfeld
Sun Microsystems
E-Mail: sommerfeld@east.sun.com
```

Appendix D. DNS Locate Specification

The DNS Locate specification is currently still an IETF draft. This document complies only with the version of the DNS Locate draft that is listed in this appendix. The PacketCable security team will continue to track progress of the DNS Locate draft through the IETF.

```
INTERNET-DRAFT                                          Ken Hornstein
<draft-ietf-cat-krb-dns-locate-02.txt>                            NRL
March 10, 2000                                         Jeffrey Altman
Expires: September 10, 2000                       Columbia University


          Distributing Kerberos KDC and Realm Information with DNS


Status of this Memo

   This document is an Internet-Draft and is in full conformance with
   all provisions of Section 10 of RFC2026.

   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.

   Distribution of this memo is unlimited.  It is filed as <draft-ietf-
   cat-krb-dns-locate-02.txt>, and expires on September 10, 2000.  Please
   send comments to the authors.

Abstract

   Neither the Kerberos V5 protocol [RFC1510] nor the Kerberos V4 proto-
   col [RFC????] describe any mechanism for clients to learn critical
   configuration information necessary for proper operation of the pro-
   tocol.  Such information includes the location of Kerberos key dis-
   tribution centers or a mapping between DNS domains and Kerberos
   realms.

   Current Kerberos implementations generally store such configuration
   information in a file on each client machine.  Experience has shown
   this method of storing configuration information presents problems
   with out-of-date information and scaling problems, especially when


Hornstein, Altman                                            [Page 1]
```

using cross-realm authentication.

This memo describes a method for using the Domain Name System [RFC1035] for storing such configuration information. Specifically, methods for storing KDC location and hostname/domain name to realm mapping information are discussed.

DNS vs. Kerberos - Case Sensitivity of Realm Names

In Kerberos, realm names are case sensitive. While it is strongly encouraged that all realm names be all upper case this recommendation has not been adopted by all sites. Some sites use all lower case names and other use mixed case. DNS on the other hand is case insensitive for queries but is case preserving for responses to TXT queries. Since "MYREALM", "myrealm", and "MyRealm" are all different it is necessary that the DNS entries be distinguishable.

Since the recommend realm names are all upper case, we will not require any quoting to be applied to upper case names. If the realm name contains lower case characters each character is to be quoted by a '=' character. So "MyRealm" would be represented as "M=yR=e=a=l=m" and "myrealm" as "=m=y=r=e=a=l=m". If the realm name contains the '=' character it will be represented as "==".

Overview - KDC location information

KDC location information is to be stored using the DNS SRV RR [RFC 2052]. The format of this RR is as follows:

Service.Proto.Realm TTL Class SRV Priority Weight Port Target

The Service name for Kerberos is always "_kerberos".

The Proto can be either "_udp" or "_tcp". If these records are to be used, a "_udp" record MUST be included. If the Kerberos implementation supports TCP transport, a "_tcp" record SHOULD be included.

The Realm is the Kerberos realm that this record corresponds to.

TTL, Class, SRV, Priority, Weight, Port, and Target have the standard meaning as defined in RFC 2052.

Example - KDC location information

These are DNS records for a Kerberos realm ASDF.COM. It has two Kerberos servers, kdc1.asdf.com and kdc2.asdf.com. Queries should be directed to kdc1.asdf.com first as per the specified priority.

Weights are not used in these records.

```
_kerberos._udp.ASDF.COM.        IN      SRV     0 0 88 kdc1.asdf.com.
_kerberos._udp.ASDF.COM.        IN      SRV     1 0 88 kdc2.asdf.com.
```

Overview - Kerberos password changing server location information

Kerberos password changing server [KERB-CHG] location is to be stored using the DNS SRV RR [RFC 2052]. The format of this RR is as follows:

```
Service.Proto.Realm TTL Class SRV Priority Weight Port Target
```

The Service name for the password server is always "_kpasswd".

The Proto MUST be "_udp".

The Realm is the Kerberos realm that this record corresponds to.

TTL, Class, SRV, Priority, Weight, Port, and Target have the standard meaning as defined in RFC 2052.

Overview - Kerberos admin server location information

Kerberos admin location information is to be stored using the DNS SRV RR [RFC 2052]. The format of this RR is as follows:

```
Service.Proto.Realm TTL Class SRV Priority Weight Port Target
```

The Service name for the admin server is always "_kerberos-adm".

The Proto can be either "_udp" or "_tcp". If these records are to be used, a "_tcp" record MUST be included. If the Kerberos admin implementation supports UDP transport, a "_udp" record SHOULD be included.

The Realm is the Kerberos realm that this record corresponds to.

TTL, Class, SRV, Priority, Weight, Port, and Target have the standard meaning as defined in RFC 2052.

Note that there is no formal definition of a Kerberos admin protocol, so the use of this record is optional and implementation-dependent.

Example - Kerberos administrative server location information

These are DNS records for a Kerberos realm ASDF.COM. It has one administrative server, kdc1.asdf.com.

```
   _kerberos-adm._tcp.ASDF.COM.    IN      SRV     0 0 88 kdc1.asdf.com.
```

Overview - Hostname/domain name to Kerberos realm mapping

Information on the mapping of DNS hostnames and domain names to Kerberos realms is stored using DNS TXT records [RFC 1035]. These records have the following format.

```
   Service.Name TTL Class TXT Realm
```

The Service field is always "_kerberos", and prefixes all entries of this type.

The Name is a DNS hostname or domain name. This is explained in greater detail below.

TTL, Class, and TXT have the standard DNS meaning as defined in RFC 1035.

The Realm is the data for the TXT RR, and consists simply of the Kerberos realm that corresponds to the Name specified.

When a Kerberos client wishes to utilize a host-specific service, it will perform a DNS TXT query, using the hostname in the Name field of the DNS query. If the record is not found, the first label of the name is stripped and the query is retried.

Compliant implementations MUST query the full hostname and the most specific domain name (the hostname with the first label removed). Compliant implementations SHOULD try stripping all subsequent labels until a match is found or the Name field is empty.

Example - Hostname/domain name to Kerberos realm mapping

For the previously mentioned ASDF.COM realm and domain, some sample records might be as follows:

```
   _kerberos.asdf.com.             IN      TXT     "ASDF.COM"
   _kerberos.mrkserver.asdf.com.   IN      TXT     "MARKETING.ASDF.COM"
   _kerberos.salesserver.asdf.com. IN      TXT     "SALES.ASDF.COM"
```

Let us suppose that in this case, a Kerberos client wishes to use a Kerberized service on the host foo.asdf.com. It would first query:

```
   _kerberos.foo.asdf.com. IN TXT
```

Finding no match, it would then query:

```
_kerberos.asdf.com. IN TXT
```

And find an answer of ASDF.COM. This would be the realm that foo.asdf.com resides in.

If another Kerberos client wishes to use a Kerberized service on the host salesserver.asdf.com, it would query:

```
_kerberos.salesserver.asdf.com IN TXT
```

And find an answer of SALES.ASDF.COM.

Security considerations

As DNS is deployed today, it is an unsecure service. Thus the information returned by it cannot be trusted.

Current practice for REALM to KDC mapping is to use hostnames to indicate KDC hosts (stored in some implementation-dependent location, but generally a local config file). These hostnames are vulnerable to the standard set of DNS attacks (denial of service, spoofed entries, etc). The design of the Kerberos protocol limits attacks of this sort to denial of service. However, the use of SRV records does not change this attack in any way. They have the same vulnerabilities that already exist in the common practice of using hostnames for KDC locations.

Current practice for HOSTNAME to REALM mapping is to provide a local configuration of mappings of hostname or domain name to realm which are then mapped to KDCs. But this again is vulnerable to spoofing via CNAME records that point to hosts in other domains. This has the same effect as when a TXT record is spoofed. In a realm with no cross-realm trusts this is a DoS attack. However, when cross-realm trusts are used it is possible to redirect a client to use a compromised realm.

This is not an exploit of the Kerberos protocol but of the Kerberos trust model. The same can be done to any application that must resolve the hostname in order to determine which domain a non-FQDN belongs to.

Implementations SHOULD provide a way of specifying this information locally without the use of DNS. However, to make this feature worthwhile a lack of any configuration information on a client should be interpreted as permission to use DNS.

```
RFC DRAFT                                                March 10, 2000

Expiration

   This Internet-Draft expires on September 10, 2000.

References


   [RFC1510]
        The Kerberos Network Authentication System; Kohl, Newman; Sep-
        tember 1993.

   [RFC1035]
        Domain Names - Implementation and Specification; Mockapetris;
        November 1987

   [RFC2782]
        A DNS RR for specifying the location of services (DNS SRV); Gul-
        brandsen, Vixie; February 2000

   [KERB-CHG]
        Kerberos Change Password Protocol; Horowitz;
        ftp://ds.internic.net/internet-drafts/draft-ietf-cat-kerb-chg-
        password-02.txt

Authors' Addresses

   Ken Hornstein
   US Naval Research Laboratory
   Bldg A-49, Room 2
   4555 Overlook Avenue
   Washington DC  20375 USA

   Phone: +1 (202) 404-4765
   EMail: kenh@cmf.nrl.navy.mil

   Jeffrey Altman
   The Kermit Project
   Columbia University
   612 West 115th Street #716
   New York NY 10025-7799 USA

   Phone: +1 (212) 854-1344
   EMail: jaltman@columbia.edu




Hornstein, Altman                                              [Page 6]
```

Appendix E. 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 F. Acknowledgements

On behalf of CableLabs and its participating member companies, I 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)

Michael Thomas, Peter Grossman, Jonathan Trostle, Jan Vilhuber and Flemming Andreasen (Cisco)

Jeff Carr and Eugene Nechamkin (Broadcom)

Mike St. Johns (Excite@Home)

Matthew Broda and Louis LeVay (Nortel Networks)

Doc Evans (Cadant/Lucent)

Matt Hur and Mike Froh (CyberSafe)

Roy Spitzer, Satish Kumar and Manjuprakish Rao (Telogy Networks)

Derek Atkins (Telcordia)

Tom Brown (Ericsson)

Ali Negahdar, Bill Tang, Rick Morris and Jiri Matousek (Arris Interactive)

Bill Aiello, Bill Marshall, Gus De Los Reyes, Chris Melle and Steve Bellovin (AT&T)

Matt Osman, CableLabs

Particular thanks are given to Sasha Medvinsky and Mike Thomas for their special dedication to this project and Nancy Davoust for the project and technical leadership of this effort for the past year.

Dave Maxwell, CableLabs