Network Working Group
Request for Comments: 4535
Category: Standards Track

H. Harney
U. Meth
A. Colegrove
SPARTA, Inc.
G. Gross
IdentAware

June 2006

GSAKMP: Group Secure Association Key Management Protocol

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document specifies the Group Secure Association Key Management Protocol (GSAKMP). The GSAKMP provides a security framework for creating and managing cryptographic groups on a network. It provides mechanisms to disseminate group policy and authenticate users, rules to perform access control decisions during group establishment and recovery, capabilities to recover from the compromise of group members, delegation of group security functions, and capabilities to destroy the group. It also generates group keys.

Table of Contents

List of Figures

List of Tables

1. Introduction

GSAKMP provides policy distribution, policy enforcement, key distribution, and key management for cryptographic groups. Cryptographic groups all share a common key (or set of keys) for data processing. These keys all support a system-level security policy so that the cryptographic group can be trusted to perform security-relevant services.

The ability of a group of entities to perform security services requires that a Group Secure Association (GSA) be established. A GSA ensures that there is a common "group-level" definition of security policy and enforcement of that policy. The distribution of cryptographic keys is a mechanism utilizing the group-level policy enforcements.

1.1. GSAKMP Overview

Protecting group information requires the definition of a security policy and the enforcement of that policy by all participating parties. Controlling dissemination of cryptographic key is the primary mechanism to enforce the access control policy. It is the primary purpose of GSAKMP to generate and disseminate a group key in a secure fashion.

GSAKMP separates group security management functions and responsibilities into three major roles:1) Group Owner, 2) Group Controller Key Server, and 3) Group Member. The Group Owner is responsible for creating the security policy rules for a group and expressing these in the policy token. The Group Controller Key Server (GC/KS) is responsible for creating and maintaining the keys and enforcing the group policy by granting access to potential Group Members (GMs) in accordance with the policy token. To enforce a group's policy, the potential Group Members need to have knowledge of the access control policy for the group, an unambiguous identification of any party downloading keys to them, and verifiable chains of authority for key download. In other words, the Group Members need to know who potentially will be in the group and to verify that the key disseminator is authorized to act in that capacity.

In order to establish a Group Secure Association (GSA) to support these activities, the identity of each party in the process MUST be unambiguously asserted and authenticated. It MUST also be verified that each party is authorized, as defined by the policy token, to function in his role in the protocol (e.g., GM or GC/KS).

The security features of the establishment protocol for the GSA include

- Group policy identification

- Group policy dissemination

- GM to GC/KS SA establishment to protect data

- Access control checking

GSAKMP provides mechanisms for cryptographic group creation and management. Other protocols may be used in conjunction with GSAKMP to allow various applications to create functional groups according to their application-specific requirements. For example, in a small-scale video conference, the organizer might use a session invitation protocol like SIP [RFC3261] to transmit information about the time of the conference, the address of the session, and the formats to be used. For a large-scale video transmission, the organizer might use a multicast announcement protocol like SAP [RFC2974].

This document describes a useful default set of security algorithms and configurations, Security Suite 1. This suite allows an entire set of algorithms and settings to be described to prospective group members in a concise manner. Other security suites MAY be defined as needed and MAY be disseminated during the out-of-band announcement of a group.

Distributed architectures support large-scale cryptographic groups. Secure distributed architectures require authorized delegation of GSA actions to network resources. The fully specified policy token is the mechanism to facilitate this authorization. Transmission of this policy token to all joining GMs allows GSAKMP to securely support distributed architectures and multiple data sources.

Many-to-many group communications require multiple data sources. Multiple data sources are supported because the inclusion of a policy token and policy payloads allow group members to review the group access control and authorization parameters. This member review process gives each member (each potential source of data) the ability to determine if the group provides adequate protection for member data.

1.2. Document Organization

The remainder of this document is organized as follows:Section 2 presents the terminology and concepts used to present the requirements of this protocol. Section 3 outlines the security considerations with respect to GSAKMP. Section 4 defines the architecture of GSAKMP. Section 5 describes the group management life cycle. Section 6 describes the Security Suite Definition. Section 7 presents the message types and formats used during each phase of the life cycle. Section 8 defines the state diagram for the protocol.

2. Terminology

The following terminology is used throughout this document.

Requirements Terminology: Keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in [RFC2119].

Certificate: A data structure used to verifiably bind an identity to a cryptographic key (e.g., X.509v3).

Compromise Recovery: The act of recovering a secure operating state after detecting that a group member cannot be trusted. This can be accomplished by rekey.

Cryptographic Group: A set of entities sharing or desiring to share a GSA.

Group Controller Key Server (GC/KS): A group member with authority to perform critical protocol actions including creating and distributing keys and building and maintaining the rekey structures. As the group evolves, it MAY become desirable to have multiple controllers perform these functions.

Group Member (GM): A Group Member is any entity with access to the group keys. Regardless of how a member becomes a part of the group or how the group is structured, GMs will perform the following actions:

- Authenticate and validate the identities and the authorizations of entities performing security-relevant actions
- Accept group keys from the GC/KS
- Request group keys from the GC/KS

- Enforce the cooperative group policies as stated in the group policy token

- Perform peer review of key management actions

- Manage local key

Group Owner (GO): A Group Owner is the entity authorized for generating and modifying an authenticatable policy token for the group, and notifying the GC/KS to start the group.

Group Policy: The Group Policy completely describes the protection mechanisms and security-relevant behaviors of the group. This policy MUST be commonly understood and enforced by the group for coherent secure operations.

Group Secure Association (GSA): A GSA is a logical association of users or hosts that share cryptographic key(s). This group may be established to support associations between applications or communication protocols.

Group Traffic Protection Key (GTPK): The key or keys created for protecting the group data.

Key Datum: A single key and its associated attributes for its usage.

Key Encryption Key (KEK): Key used in an encryption mechanism for wrapping another key.

Key Handle: The identifier of a particular instance or version of a key.

Key ID: The identifier for a key that MUST stay static throughout the life cycle of this key.

Key Package: Type/Length/Data format containing a Key Datum.

Logical Key Hierarchy (LKH) Array: The group of keys created to facilitate the LKH compromise recovery methodology.

Policy Token (PT): The policy token is a data structure used to disseminate group policy and the mechanisms to enforce it. The policy token is issued and signed by an authorized Group Owner. Each member of the group MUST verify the token, meet the group join policy, and enforce the policy of the group (e.g., encrypt application data with a specific algorithm). The group policy token will contain a variety of information including:

- GSAKMP protocol version
- Key creation method
- Key dissemination policy
- Access control policy
- Group authorization policy
- Compromise recovery policy
- Data protection mechanisms

Rekey: The act of changing keys within a group as defined by policy.

Rekey Array: The construct that contains all the rekey information for a particular member.

Rekey Key: The KEK used to encrypt keys for a subset of the group.

Subordinate Group Controller Key Server (S-GC/KS): Any group member having the appropriate processing and trust characteristics, as defined in the group policy, that has the potential to act as a S-GC/KS. This will allow the group processing and communication requirements to be distributed equitably throughout the network (e.g., distribute group key). The optional use of GSAKMP with Subordinate Group Controller Key Servers will be documented in a separate paper.

Wrapping KeyID: The Key ID of the key used to wrap a Key Package.

Wrapping Key Handle: The key handle of the key used to wrap the Key Package.

3. Security Considerations

In addition to the specification of GSAKMP itself, the security of an implemented GSAKMP system is affected by supporting factors. These are discussed here.

3.1. Security Assumptions

The following assumptions are made as the basis for the security discussion:

1. GSAKMP assumes its supporting platform can provide the process and data separation services at the appropriate assurance level to support its groups.

2. The key generation function of the cryptographic engine will only generate strong keys.

3. The security of this protocol is critically dependent on the randomness of the randomly chosen parameters. These should be generated by a strong random or properly seeded pseudo-random source [RFC4086].

4. The security of a group can be affected by the accuracy of the system clock. Therefore, GSAKMP assumes that the system clock is close to correct time. If a GSAKMP host relies on a network time service to set its local clock, then that protocol must be secure against attackers. The maximum allowable clock skew across the group membership is policy configurable, with a default of 5 minutes.

5. As described in the message processing section, the use of the nonce value used for freshness along with a signature is the mechanism used to foil replay attacks. In any use of nonces, a core requirement is unpredictability of the nonce, from an attacker's viewpoint. The utility of the nonce relies on the inability of an attacker either to reuse old nonces or to predict the nonce value.

6. GSAKMP does not provide identity protection.

7. The group's multicast routing infrastructure is not secured by GSAKMP, and therefore it may be possible to create a multicast flooding denial of service attack using the multicast application's data stream. Either an insider (i.e., a rogue GM) or a non-member could direct the multicast routers to spray data at a victim system.

8. The compromise of a S-GC/KS forces the re-registration of all GMs under its control. The GM recognizes this situation by finding the S-GC/KS's certificate on a CRL as supplied by a service such as LDAP.

9. The compromise of the GO forces termination of the group. The GM recognizes this situation by finding the GO's certificate on a Certificate Revocation List (CRL) as supplied by a service such as LDAP.

3.2. Related Protocols

GSAKMP derives from two (2) existing protocols: ISAKMP [RFC2408] and FIPS Pub 196 [FIPS196]. In accordance with Security Suite 1, GSAKMP implementations MUST support the use of Diffie-Hellman key exchange [DH77] for two-party key creation and MAY use Logical Key Hierarchy (LKH) [RFC2627] for rekey capability. The GSAKMP design was also influenced by the following protocols: [HHMCD01], [RFC2093], [RFC2094], [BMS], and [RFC2412].

3.2.1. ISAKMP

ISAKMP provides a flexible structure of chained payloads in support of authenticated key exchange and security association management for pairwise communications. GSAKMP builds upon these features to provide policy enforcement features in support of diverse group communications.

3.2.2. FIPS Pub 196

FIPS Pub 196 provides a mutual authentication protocol.

3.2.3. LKH

When group policy dictates that a recovery of the group security is necessary after the discovery of the compromise of a GM, then GSAKMP relies upon a rekey capability (i.e., LKH) to enable group recovery after a compromise [RFC2627]. This is optional since in many instances it may be better to destroy the compromised group and rebuild a secure group.

3.2.4. Diffie-Hellman

A Group may rely upon two-party key creation mechanisms, i.e., Diffie-Hellman, to protect sensitive data during download.

The information in this section borrows heavily from [IKEv2], as this protocol has already worked through similar issues and GSAKMP is using the same security considerations for its purposes. This section will contain paraphrased sections of [IKEv2] modified for GSAKMP as appropriate.

The strength of a key derived from a Diffie-Hellman exchange using specific p and g values depends on the inherent strength of the values, the size of the exponent used, and the entropy provided by the random number generator used. A strong random number generator combined with the recommendations from [RFC3526] on Diffie-Hellman exponent size is recommended as sufficient. An implementation should make note of this conservative estimate when establishing policy and negotiating security parameters.

Note that these limitations are on the Diffie-Hellman values themselves. There is nothing in GSAKMP that prohibits using stronger values, nor is there anything that will dilute the strength obtained from stronger values. In fact, the extensible framework of GSAKMP encourages the definition of more Security Suites.

It is assumed that the Diffie-Hellman exponents in this exchange are erased from memory after use. In particular, these exponents MUST NOT be derived from long-lived secrets such as the seed to a pseudo-random generator that is not erased after use.

3.3. Denial of Service (DoS) Attack

This GSAKMP specification addresses the mitigation for a distributed IP spoofing attack (a subset of possible DoS attacks) in Section 5.2.2, "Cookies: Group Establishment with Denial of Service Protection".

3.4. Rekey Availability

In addition to GSAKMP's capability to do rekey operations, GSAKMP MUST also have the capability to make this rekey information highly available to GMs. The necessity of GMs receiving rekey messages requires the use of methods to increase the likelihood of receipt of rekey messages. These methods MAY include multiple transmissions of the rekey message, posting of the rekey message on a bulletin board, etc. Compliant GSAKMP implementations supporting the optional rekey capability MUST support retransmission of rekey messages.

3.5. Proof of Trust Hierarchy

As defined by [HCM], security group policy MUST be defined in a verifiable manner. GSAKMP anchors its trust in the creator of the group, the GO.

The policy token explicitly defines all the parameters that create a secure verifiable infrastructure. The GSAKMP Policy Token is issued and signed by the GO. The GC/KS will verify it and grant access to GMs only if they meet the rules of the policy token. The new GMs will accept access only if 1) the token verifies, 2) the GC/KS is an authorized disseminator, and 3) the group mechanisms are acceptable for protecting the GMs data.

4. Architecture

This architecture presents a trust model for GSAKMP and a concept of operations for establishing a trusted distributed infrastructure for group key and policy distribution.

GSAKMP conforms to the IETF MSEC architectural concepts as specified in the MSEC Architecture document [RFC3740]. GSAKMP uses the MSEC components to create a trust model for operations that implement the security principles of mutual suspicion and trusted policy creation authorities.

4.1. Trust Model

4.1.1. Components

The trust model contains four key components:

- Group Owner (GO),
- Group Controller Key Server (GC/KS),
- Subordinate GC/KS (S-GC/KS), and
- Group Member (GM).

The goal of the GSAKMP trust model is to derive trust from a common trusted policy creation authority for a group. All security-relevant decisions and actions implemented by GSAKMP are based on information that ultimately is traceable to and verified by the trusted policy creation authority. There are two trusted policy creation authorities for GSAKMP: the GO (policy creation authority) and the PKI root that allows us to verify the GO.

4.1.2. GO

The GO is the policy creation authority for the group. The GO has a well-defined identity that is relevant to the group. That identity can be of a person or of a group-trusted component. All potential entities in the group have to recognize the GO as the individual with authority to specify policy for the group.

The policy reflects the protection requirements of the data in a group. Ultimately, the data and the application environment drives the security policy for the group.

The GO has to determine the security rules and mechanisms that are appropriate for the data being protected by the group keys. All this information is captured in a policy token (PT). The GO creates the PT and signs it.

4.1.3. GC/KS

The GC/KS is authorized to perform several functions: key creation, key distribution, rekey, and group membership management.

As the key creation authority, the GC/KS will create the set of keys for the group. These keys include the Group Traffic Protection Keys (GTPKs) and first-tier rekey keys. There may be second-tier rekey trees if a distributed rekey management structure is required for the group.

As the key distribution (registration) authority, it has to notify the group of its location for registration services. The GC/KS will have to enforce key access control as part of the key distribution and registration processes.

As the group rekey authority, it performs rekey in order to change the group's GTPK. Change of the GTPK limits the exposure of data encrypted with any single GTPK.

Finally, as the group membership management authority, the GC/KS can manage the group membership (registration, eviction, de-registration, etc.). This may be done in part by using a key tree approach, such as Logical Key Hierarchies (LKH), as an optional approach.

4.1.4. Subordinate GC/KS

A subordinate GC/KS is used to distribute the GC/KS functionality across multiple entities. The S-GC/KS will have all the authorities of the GC/KS except one: it will not create the GTPK. It is assumed here that the group will transmit data with a single GTPK at any one time. This GTPK comes from the GC/KS.

Note that relative to the GC/KS, the S-GC/KS is responsible for an additional security check: the S-GC/KS must register as a member with the GC/KS, and during that process it has to verify the authority of the GC/KS.

4.1.5. GM

The GM has two jobs: to make sure all security-relevant actions are authorized and to use the group keys properly. During the registration process, the GM will verify that the PT is signed by a recognized GO. In addition, it will verify that the GC/KS or S-GC/KS engaged in the registration process is authorized, as specified in the PT. If rekey and new PTs are distributed to the group, the GM will verify that they are proper and all actions are authorized.

The GM is granted access to group data through receipt of the group keys This carries along with it a responsibility to protect the key from unauthorized disclosure.

GSAKMP does not offer any enforcement mechanisms to control which GMs are multicast speakers at a given moment. This policy and its enforcement depend on the multicast application and its protocols. However, GSAKMP does allow a group to have one of three Group Security Association multicast speaker configurations:

- There is a single GM authorized to be the group's speaker. There is one multicast application SA allocated by the GO in support of that speaker. The PT initializes this multicast application SA and identifies the GM that has been authorized to be speaker. All GMs share a single TPK with that GM speaker. Sequence number checking for anti-replay protection is feasible and enabled by default. This is the default group configuration. GSAKMP implementations MUST support this configuration.

- The GO authorizes all of the GMs to be group speakers. The GO allocates one multicast application SA in support of these speakers. The PT initializes this multicast application SA and indicates that any GM can be a speaker. All of the GMs share a single GTPK and other SA state information. Consequently, some SA security features such as sequence number checking for anti-replay

protection cannot be supported by this configuration. GSAKMP implementations MUST support this group configuration.

- The GO authorizes a subset of the GMs to be group speakers (which may be the subset composed of all GMs). The GO allocates a distinct multicast application SA for each of these speakers. The PT identifies the authorized speakers and initializes each of their multicast application Security Associations. The speakers still share a common TPK across their SA, but each speaker has a separate SA state information instance at every peer GM. Consequently, this configuration supports SA security features, such as sequence number checking for anti-replay protection, or source authentication mechanisms that require per-speaker state at the receiver. The drawback of this configuration is that it does not scale to a large number of speakers. GSAKMP implementations MAY support this group configuration.

4.1.6. Assumptions

The assumptions for this trust model are that:

- the GCKS is never compromised,
- the GO is never compromised,
- the PKI, subject to certificate validation, is trustworthy,
- The GO is capable of creating a security policy to meet the demands of the group,
- the compromises of a group member will be detectable and reported to the GO in a trusted manner,
- the subsequent recovery from a compromise will deny inappropriate access to protected data to the compromised member,
- no security-relevant actions depend on a precise network time,
- there are confidentiality, integrity, multicast source authentication, and anti-replay protection mechanisms for all GSAKMP control messages.

4.2. Rule-Based Security Policy

The trust model for GSAKMP revolves around the definition and enforcement of the security policy. In fact, the use of the key is only relevant, in a security sense, if it represents the successful enforcement of the group security policy.

Group operations lend themselves to rule-based security policy. The need for distribution of data to many endpoints often leads to the defining of those authorized endpoints based on rules. For example, all IETF attendees at a given conference could be defined as a single group.

If the security policy rules are to be relevant, they must be coupled with validation mechanisms. The core principle here is that the level of trust one can afford a security policy is exactly equal to the level of trust one has in the validation mechanism used to prove that policy. For example, if all IETF attendees are allowed in, then they could register their identity from their certificate upon check-in to the meetings. That certificate is issued by a trusted policy creation authority (PKI root) that is authorized to identify someone as an IETF attendee. The GO could make admittance rules to the IETF group based on the identity certificates issued from trusted PKIs.

In GSAKMP, every security policy rule is coupled with an explicit validation mechanism. For interoperability considerations, GSAKMP requires that its supporting PKI implementations MUST be compliant to RFC 3280.

If a GM's public key certificate is revoked, then the entity that issues that revocation SHOULD signal the GO, so that the GO can expel that GM. The method that signals this event to the GO is not standardized by this specification.

A direct mapping of rule to validation mechanism allows the use of multiple rules and PKIs to create groups. This allows a GO to define a group security policy that spans multiple PKI domains, each with its own Certificate Authority public key certificate.

4.2.1. Access Control

The access control policy for the group keys is equivalent to the access control policy for the multicast application data the keys are protecting.

In a group, each data source is responsible for ensuring that the access to the source's data is appropriate. This implies that every data source should have knowledge of the access control policy for the group keys.

In the general case, GSAKMP offers a suite of security services to its applications and does not prescribe how they use those services.

GSAKMP supports the creation of GSAs with multiple data sources. It also supports architectures where the GC/KS is not itself a data source. In the multiple data source architectures GSAKMP requires that the access control policy is precisely defined and distributed to each data source. The reference for this data structure is the GSAKMP Policy Token [RFC4534].

4.2.2. Authorizations for Security-Relevant Actions

A critical aspect of the GSAKMP trust model is the authorization of security-relevant actions. These include download of group key, rekey, and PT creation and updates. These actions could be used to disrupt the secure group, and all entities in the group must verify that they were instigated by authorized entities within the group.

4.3. Distributed Operation

Scalability is a core feature of GSAKMP. GSAKMP's approach to scalable operations is the establishment of S-GC/KSes. This allows the GSAKMP systems to distribute the workload of setting up and managing very large groups.

Another aspect of distributed S-GC/KS operations is the enabling of local management authorities. In very large groups, subordinate enclaves may be best suited to provide local management of the enclaves' group membership, due to a direct knowledge of the group members.

One of the critical issues involved with distributed operation is the discovery of the security infrastructure location and security suite. Many group applications that have dynamic interactions must "find" each other to operate. The discovery of the security infrastructure is just another piece of information that has to be known by the group in order to operate securely.

There are several methods for infrastructure discovery:

- Announcements

- Anycast

- Rendezvous points / Registration

One method for distributing the security infrastructure location is to use announcements. The SAP is commonly used to announce the existence of a new multicast application or service. If an

application uses SAP [RFC2974] to announce the existence of a service on a multicast channel, that service could be extended to include the security infrastructure location for a particular group.

Announcements can also be used by GSAKMP in one of two modes: expanding ring searches (ERSes) of security infrastructure and ERSes for infrastructure discovery. In either case, the GSAKMP would use a multicast broadcast that would slowly increase in its range by incremental multicast hops. The multicast source controls the packet's multicast range by explicitly setting its Time To Live count.

An expanding ring announcement operates by the GC/KS announcing its existence for a particular group. The number of hops this announcement would travel would be a locally configured number. The GMs would listen on a well-known multicast address for GC/KSes that provide service for groups of interest. If multiple GC/KSes are found that provide service, then the GM would pick the closest one (in terms of multicast hops). The GM would then send a GSAKMP Request to Join message (RTJ) to the announced GC/KS. If the announcement is found to be spurious, then that is reported to the appropriate management authorities. The ERA concept is slightly different from SAP in that it could occur over the data channel multicast address, instead of a special multicast address dedicated for the SAP service.

An expanding ring search operates in the reverse order of the ERA. In this case, the GM is the announcing entity. The (S-)GC/KSes listen for the requests for service, specifically the RTJ. The (S-)GC/KS responds to the RTJ. If the GM receives more than one response, it would either ignore the responses or send NACKs based on local configuration.

Anycast is a service that is very similar to ERS. It also can be used to provide connection to the security infrastructure. In this case, the GM would send the RTJ to a well-known service request address. This anycast service would route the RTJ to an appropriate GC/KS. The anycast service would have security infrastructure and network connectivity knowledge to facilitate this connection.

Registration points can be used to distribute many group-relevant data, including security infrastructure. Many group applications rely on well-known registration points to advertise the availability of groups. There is no reason that GSAKMP could not use the same approach for advertising the existence and location of the security infrastructure. This is a simple process if the application being supported already supports registration. The GSAKMP infrastructure can always provide a registration site if the existence of this

security infrastructure discovery hub is needed. The registration of S-GC/KSes at this site could be an efficient way to allow GM registration.

GSAKMP infrastructure discovery can use whatever mechanism suits a particular multicast application's requirements, including mechanisms that have not been discussed by this architecture. However, GSAKMP infrastructure discovery is not standardized by this version of the GSAKMP specification.

4.4. Concept of Operation

This concept of operation shows how the different roles in GSAKMP interact to set up a secure group. This particular concept of operation focuses on a secure group that utilizes the distributed key dissemination services of the S-GC/KS.

4.4.1. Assumptions

The most basic assumption here is that there is one or more trustworthy PKIs for the group. That trusted PKI will be used to create and verify security policy rules.

There is a GO that all GMs recognize as having group policy creation authority. All GM must be securely pre-configured to know the GO public key.

All GMs have access to the GO PKI information, both the trusted anchor public keys and the certificate path validation rules.

There is sufficient connectivity between the GSAKMP entities.

- The registration SA requires that GM can connect to the GC/KS or S-GC/KS using either TCP or UDP.

- The Rekey SA requires that the data-layer multicast communication service be available. This can be multicast IP, overlay networks using TCP, or NAT tunnels.

- GSAKMP can support many different data-layer secure applications, each with unique connectivity requirements.

4.4.2. Creation of a Policy Token

The GO creates and signs the policy token for a group. The policy token contains the rules for access control and authorizations for a particular group.

The PT consists of the following information:

- Identification: This allows an unambiguous identification of the PT and the group.

- Access Control Rules: These rules specify who can have access to the group keys.

- Authorization Rules: These rules specify who can be a S-GC/KS.

- Mechanisms: These rules specify the security mechanisms that will be used by the group. This is necessary to ensure there is no weak link in the group security profile. For example, for IPsec, this could include SPD/SAD configuration data.

- Source authentication of the PT to the GO: The PT is a CMS signed object, and this allows all GMs to verify the PT.

4.4.3. Creation of a Group

The PT is sent to a potential GC/KS. This can occur in several ways, and the method of transmittal is outside the scope of GSAKMP. The potential GC/KS will verify the GO signature on the PT to ensure that it comes from a trusted GO. Next, the GC/KS will verify that it is authorized to become the GC/KS, based on the authorization rules in the PT. Assuming that the GC/KS trusts the PT, is authorized to be a GC/KS, and is locally configured to become a GC/KS for a given group and the GO, then the GC/KS will create the keys necessary to start the group. The GC/KS will take whatever action is necessary (if any) to advertise its ability to distribute key for the group. The GC/KS will then listen for RTJs.

The PT has a sequence number. Every time a PT is distributed to the group, the group members verify that the sequence number on the PT is increasing. The PT lifetime is not limited to a particular time interval, other than by the lifetimes imposed by some of its attributes (e.g., signature key lifetime). The current PT sequence number is downloaded to the GM in the "Key Download" message. Also, to avoid replay attacks, this sequence number is never reset to a lower value (i.e., rollover to zero) as long as the group identifier remains valid and in use. The GO MUST preserve this sequence number across re-boots.

4.4.4. Discovery of GC/KS

Potential GMs will receive notice of the new group via some mechanism: announcement, Anycast, or registration look-up. The GM will send an RTJ to the GC/KS.

4.4.5. GC/KS Registration Policy Enforcement

The GC/KS may or may not require cookies, depending on the DoS environment and the local configuration.

Once the RTJ has been received, the GC/KS will verify that the GM is allowed to have access to the group keys. The GC/KS will then verify the signature on the RTJ to ensure it was sent by the claimed identity. If the checks succeed, the GC/KS will ready a Key Download message for the GM. If not, the GC/KS can notify the GM of a non-security-relevant problem.

4.4.6. GM Registration Policy Enforcement

Upon receipt of the Key Download message, the GM will verify the signature on the message. Then the GM will retrieve the PT from the Key Download message and verify that the GO created and signed the PT. Once the PT is verified as valid, the GM will verify that the GC/KS is authorized to distribute key for this group. Then the GM will verify that the mechanisms used in the group are available and acceptable for protection of the GMs data (assuming the GM is a data source). The GM will then accept membership in this group.

The GM will then check to see if it is allowed to be a S-GC/KS for this group. If the GM is allowed to be a S-GC/KS AND the local GM configuration allows the GM to act as a S-GC/KS for this group, then the GM changes its operating state to S-GC/KS. The GO needs to assign the authority to become a S-GC/KS in a manner that supports the overall group integrity and operations.

4.4.7. Autonomous Distributed GSAKMP Operations

In autonomous mode, each S-GC/KS operates a largely self-contained sub-group for which the Primary-GC/KS delegates the sub-group's membership management responsibility to the S-GC/KS. In general, the S-GC/KS locally handles each Group Member's registration and de-registration without any interaction with the Primary-GC/KS. Periodically, the Primary-GC/KS multicasts a Rekey Event message addressed only to its one or more S-GC/KS.

After a S-GC/KS successfully processes a Rekey Event message from the Primary-GC/KS, the S-GC/KS transmits to its sub-group its own Rekey

Event message containing a copy of the group's new GTPK and policy token. The S-GC/KS encrypts its Rekey Event message's sub-group key management information using Logical Key Hierarchy or a comparable rekey protocol. The S-GC/KS uses the rekey protocol to realize forward and backward secrecy, such that only the authorized sub-group members can decrypt and acquire access to the new GTPK and policy token. The frequency at which the Primary-GC/KS transmits a Rekey Event message is a policy token parameter.

For the special case of a S-GC/KS detecting an expelled or compromised group member, a mechanism is defined to trigger an immediate group rekey rather than wait for the group's rekey period to elapse. See below for details.

Each S-GC/KS will be registered by the GC/KS as a management node with responsibility for GTPK distribution, access control policy enforcement, LKH tree creation, and distribution of LKH key arrays. The S-GC/KS will be registered into the primary LKH tree as an endpoint. Each S-GC/KS will hold an entire LKH key array for the GC's LKH key tree.

For the purpose of clarity, the process of creating a distributed GSAKMP group will be explained in chronological order.

First, the Group Owner will create a policy token that authorizes a subset of the group's membership to assume the role of S-GC/KS.

The GO needs to ensure that the S-GC/KS rules in the policy token will be stringent enough to ensure trust in the S-GC/KSes. This policy token is handed off to the primary GC.

The GC will create the GTPK and initial LKH key tree. The GC will then wait for a potential S-GC/KS to send a Request to Join (RTJ) message.

A potential S-GC/KS will eventually send an RTJ. The GC will enforce the access control policy as defined in the policy token. The S-GC/KS will accept the role of S-GC/KS and create its own LKH key tree for its sub-group membership.

The S-GC/KS will then offer registration services for the group. There are local management decisions that are optional to control the scope of group members that can be served by a S-GC/KS. These are truly local management issues that allow the administrators of an S-GC/KS to restrict service to potential GMs. These local controls do not affect the overall group security policy, as defined in the policy token.

A potential Group Member will send an RTJ to the S-GC/KS. The S-GC/KS will enforce the entire access control policy as defined in the PT. The GM will receive an LKH key array that corresponds to the LKH tree of the S-GC/KS. The key tree generated by the S-GC/KS is independent of the key tree generated by the GC/KS; they share no common keys.

The GM then has the keys it needs to receive group traffic and be subject to rekey from the S-GC/KS. For the sake of this discussion, let's assume the GM is to be expelled from the group membership.

The S-GC/KS will receive notification that the GM is to be expelled. This mechanism is outside the scope of this protocol.

Upon notification that a GM that holds a key array within its LKH tree is to be expelled, the S-GC/KS does two things. First, the S-GC/KS initiates a de-registration exchange with the GC/KS identifying the member to be expelled. (The S-GC/KS proxies a Group Member's de-registration informing the GC/KS that the Group Member has been expelled from the group.) Second, the S-GC/KS will wait for a rekey action by the GC/KS. The immediacy of the rekey action by the GC/KS is a management decision at the GC/KS. Security is best served by quick expulsion of untrusted members.

Upon receipt of the de-registration notification from the S-GC/KS, the GC/KS will register the member to be expelled. The GC/KS will then follow group procedure for initiating a rekey action (outside the scope of this protocol). The GC/KS will communicate to the GO the expelled member's information (outside the scope of this protocol). With this information, the GO will create a new PT for the group with the expelled GM identity added to the excluded list in the group's access control rules. The GO provides this new PT to the GC/KS for distribution with the Rekey Event Message.

The GC/KS will send out a rekey operation with a new PT. The S-GC/KS will receive the rekey and process it. At the same time, all other S-GC/KSes will receive the rekey and note the excluded GM identity. All S-GC/KSes will review local identities to ensure that the excluded GM is not a local member. If it is, then the S-GC/KS will create a rekey message. The S-GC/KSes must always create a rekey message, whether or not the expelled Group Member is a member of their subtrees.

The S-GC/KS will then create a local rekey message. The S-GC/KS will send the wrapped Group TPK to all members of its local LKH tree, except the excluded member(s).

5. Group Life Cycle

The management of a cryptographic group follows a life cycle: group definition, group establishment, and security-relevant group maintenance. Group definition involves defining the parameters necessary to support a secure group, including its policy token. Group establishment is the process of granting access to new members. Security-relevant group maintenance messages include rekey, policy changes, member deletions, and group destruction. Each of these life cycle phases is discussed in the following sections.

The use and processing of the optional Vendor ID payload for all messages can be found in Section 7.10.

5.1. Group Definition

A cryptographic group is established to support secure communications among a group of individuals. The activities necessary to create a policy token in support of a cryptographic group include:

- Determine Access Policy: identify the entities that are authorized to receive the group key.

- Determine Authorization Policy: identify which entities are authorized to perform security-relevant actions, including key dissemination, policy creation, and initiation of security-management actions.

- Determine Mechanisms: define the algorithms and protocols used by GSAKMP to secure the group.

- Create Group Policy Token: format the policies and mechanisms into a policy token, and apply the GO signature.

5.2. Group Establishment

GSAKMP Group Establishment consists of three mandatory-to-implement messages: the Request to Join, the Key Download, and the Key Download Ack/Failure. The exchange may also include two OPTIONAL error messages: the Request to Join Error and the Lack_of_Ack messages. Operation using the mandatory messages only is referred to as "Terse Mode", while inclusion of the error messaging is referred to as "Verbose Mode". GSAKMP implementations MUST support Terse Mode and MAY support Verbose Mode. Group Establishment is discussed in Section 5.2.1.

A group is set in Terse or Verbose Mode by a policy token parameter. All (S-)GC/KSes in a Verbose Mode group MUST support Verbose Mode. GSAKMP allows Verbose Mode groups to have GMs that do not support Verbose Mode. Candidate GMs that do not support Verbose Mode and receive a RTJ-Error or Lack-of-Ack message must handle these messages gracefully. Additionally, a GM will not know ahead of time that it is interacting with the (S-)GC/KS in Verbose or Terse Mode until the policy token is received.

For denial of service protection, a Cookie Exchange MAY precede the Group Establishment exchange. The Cookie Exchange is described in Section 5.2.2.

Regardless of mode, any error message sent between component members indicates the first error encountered while processing the message.

5.2.1. Standard Group Establishment

After the out-of-band receipt of a policy token, a potential Group Controller Key Server (GC/KS) verifies the token and its eligibility to perform GC/KS functionality. It is then permitted to create any needed group keys and begin to establish the group.

The GSAKMP Ladder Diagram, Figure 1, illustrates the process of establishing a cryptographic group. The left side of the diagram represents the actions of the GC/KS. The right side of the diagram represents the actions of the GMs. The components of each message shown in the diagram are presented in Sections 5.2.1.1 through 5.2.1.5.

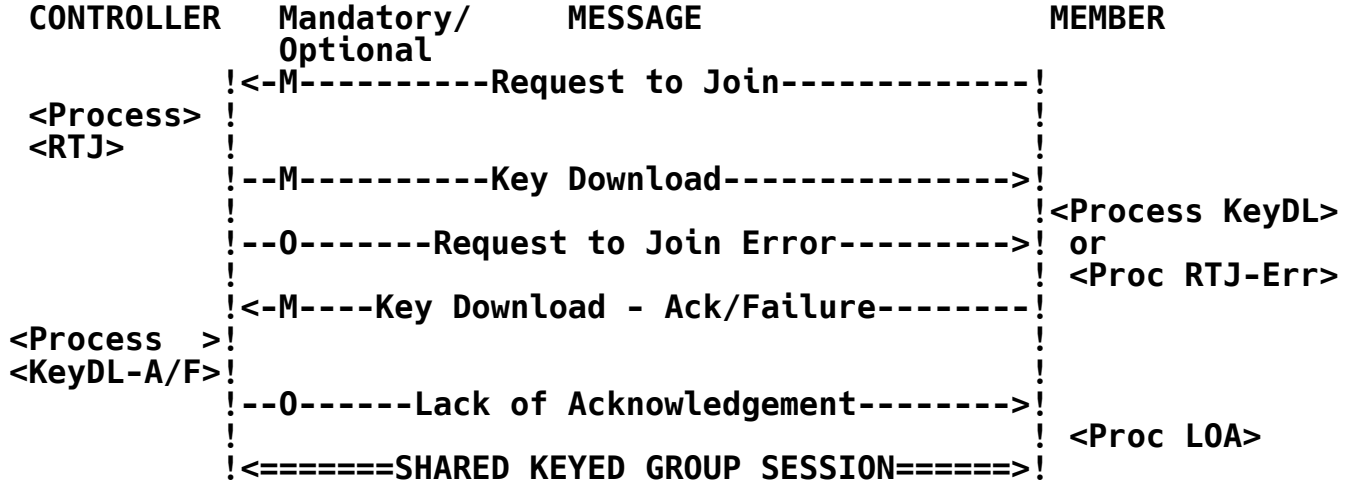

Figure 1: GSAKMP Ladder Diagram

The Request to Join message is sent from a potential GM to the GC/KS to request admission to the cryptographic group. The message contains key creation material, freshness data, an optional selection of mechanisms, and the signature of the GM.

The Key Download message is sent from the GC/KS to the GM in response to an accepted Request to Join. This GC/KS-signed message contains the identifier of the GM, freshness data, key creation material, encrypted keys, and the encrypted policy token. The policy token is used to facilitate well-ordered group creation and MUST include the group's identification, group permissions, group join policy, group controller key server identity, group management information, and digital signature of the GO. This will allow the GM to determine whether group policy is compatible with local policy.

The Request to Join Error message is sent from the GC/KS to the GM in response to an unaccepted Request to Join. This message is not signed by the GC/KS for two reasons: 1) the GM, at this point, has no knowledge of who is authorized to act as a GC/KS, and so the signature would thus be meaningless to the GM, and 2) signing responses to denied join requests would provide a denial of service potential. The message contains an indication of the error condition. The possible values for this error condition are: Invalid-Payload-Type, Invalid-Version, Invalid-Group-ID, Invalid-Sequence-ID, Payload-Malformed, Invalid-ID-Information, Invalid-Certificate, Cert-Type-Unsupported, Invalid-Cert-Authority, Authentication-Failed, Certificate-Unavailable, Unauthorized-Request, Prohibited-by-Group-Policy, and Prohibited-by-Locally-Configured-Policy.

The Key Download Ack/Failure message indicates Key Download receipt status at the GM. It is a GM-signed message containing freshness data and status.

The Lack_of_Ack message is sent from the GC/KS to the GM in response to an invalid or absent Key Download Ack/Failure message. The signed message contains freshness and status data and is used to warn the GM of impending eviction from the group if a valid Key Download Ack/Failure is not sent. Eviction means that the member will be excluded from the group after the next Rekey Event. The policy of when a particular group needs to rekey itself is stated in the policy token. Eviction is discussed further in Section 5.3.2.1.

For the following message structure sections, details about payload format and processing can be found in Section 7. Each message is identified by its exchange type in the header of the message. Nonces MUST be present in the messages unless synchronization time is available to the system.

5.2.1.1. Request to Join

The exchange type for Request to Join is eight (8).

The components of a Request to Join Message are shown in Table 1.

Table 1: Request to Join (RTJ) Message Definition

```
Message Name  : Request to Join (RTJ)
Dissection    : {HDR-GrpID, Key Creation, Nonce_I, [VendorID],
              : [Notif_Mechanism_Choices], [Notif_Cookie],
              : [Notif_IPValue]} SigM, [Cert]
Payload Types : GSAKMP Header, Key Creation, [Nonce], [Vendor
                ID], Signature, [Certificate], [Notifications]

  SigM        : Signature of Group Member
  Cert        : Necessary Certificates, zero or more
  {}SigX      : Indicates fields used in Signature
  []          : Indicate an optional data item
```

As shown by Figure 1, a potential GM MUST generate and send an RTJ message to request permission to join the group. At a minimum, the GM MUST be able to manually configure the destination for the RTJ. As defined in the dissection of the RTJ message, this message MUST contain a Key Creation payload for KEK determination. A Nonce payload MUST be included for freshness and the Nonce_I value MUST be saved for potential later use. The GC/KS will use this supplied nonce only if the policy token for this group defines the use of nonces versus synchronization time. An OPTIONAL Notification payload of type Mechanism Choices MAY be included to identify the mechanisms the GM wants to use. Absence of this payload will cause the GC/KS to select appropriate default policy-token-specified mechanisms for the Key Download.

In response, the GC/KS accepts or denies the request based on local configuration. <Process RTJ> indicates the GC/KS actions that will determine if the RTJ will be acted upon. The following checks SHOULD be performed in the order presented.

In this procedure, the GC/KS MUST verify that the message header is properly formed and confirm that this message is for this group by checking the value of the GroupID. If the header checks pass, then the identity of the sender is extracted from the Signature payload. This identity MUST be used to perform access control checks and find the GMs credentials (e.g., certificate) for message verification. It MUST also be used in the Key Download message. Then, the GC/KS will verify the signature on the message to ensure its authenticity. The

GC/KS MUST use verified and trusted authentication material from a known root. If the message signature verifies, the GC/KS then confirms that all required payloads are present and properly formatted based upon the mechanisms announced and/or requested. If all checks pass, the GC/KS will create and send the Key Download message as described in Section 5.2.1.2.

If the GM receives no response to the RTJ within the GM's locally configured timeout value, the GM SHOULD resend the RTJ message up to three (3) times.

NOTE: At any one time, a GC/KS MUST process no more than one (1) valid RTJ message from a given GM per group until its pending registration protocol exchange concludes.

If any error occurs during RTJ message processing, and the GC/KS is running in Terse Mode, the registration session MUST be terminated, and all saved state information MUST be cleared.

The OPTIONAL Notification payload of type Cookie is discussed in Section 5.2.2.

The OPTIONAL Notification payload of type IPValue may be used for the GM to convey a specific IP value to the GC/KS.

5.2.1.2. Key Download

The exchange type for Key Download is nine (9).

The components of a Key Download Message are shown in Table 2:

Table 2: Key Download (KeyDL) Message Definition

```
Message Name  : Key Download (KeyDL)
Dissection    : {HDR-GrpID, Member ID, [Nonce_R, Nonce_C], Key
                Creation, (Policy Token)*, (Key Download)*,
                [VendorID]} SigC, [Cert]
Payload Types : GSAKMP Header, Identification, [Nonce], Key
                Creation, Policy Token, Key Download, [Vendor
                ID], Signature, [Certificate]

  SigC        : Signature of Group Controller Key Server
  Cert        : Necessary Certificates, zero or more
  {}SigX      : Indicates fields used in Signature
  []          : Indicate an optional data item
  (data)*     : Indicates encrypted information
```

In response to a properly formed and verified RTJ message, the GC/KS creates and sends the KeyDL message. As defined in the dissection of the message, this message MUST contain payloads to hold the following information: GM identification, Key Creation material, encrypted policy token, encrypted key information, and signature information. If synchronized time is not available, the Nonce payloads MUST be included in the message for freshness.

If present, the nonce values transmitted MUST be the GC/KS's generated Nonce_R value and the combined Nonce_C value that was generated by using the GC/KS's Nonce_R value and the Nonce_I value received from the GM in the RTJ.

If two-party key determination is used, the key creation material supplied by the GM and/or the GC/KS will be used to generate the key. Generation of this key is dependent on the key exchange, as defined in Section 7.11, "Key Creation Payload". The policy token and key material are encrypted in the generated key.

The GM MUST be able to process the Key Download message. <Process KeyDL> indicates the GM actions that will determine how the Key Download message will be acted upon. The following checks SHOULD be performed in the order presented.

In this procedure, the GM will verify that the message header is properly formed and confirm that this message is for this group by checking the value of the GroupID. If the header checks pass, the GM MUST confirm that this message was intended for itself by comparing the Member ID in the Identification payload to its identity.

After identification confirmation, the freshness values are checked. If using nonces, the GM MUST use its saved Nonce_I value, extract the received GC/KS Nonce_R value, compute the combined Nonce_C value, and compare it to the received Nonce_C value. If not using nonces, the GM MUST check the timestamp in the Signature payload to determine if the message is new.

After freshness is confirmed, the signature MUST be verified to ensure its authenticity. The GM MUST use verified and trusted authentication material from a known root. If the message signature verifies, the key creation material is extracted from the Key Creation payload to generate the KEK. This KEK is then used to decrypt the policy token data. The signature on the policy token MUST be verified. Access control checks MUST be performed on both the GO and the GC/KS to determine both their authorities within this group. After all these checks pass, the KEK can then be used to

decrypt and process the key material from the Key Download payload. If all is successful, the GM will create and send the Key Download - Ack/Failure message as described in Section 5.2.1.4.

The Policy Token and Key Download Payloads are sent encrypted in the KEK generated by the Key Creation Payload information using the mechanisms defined in the group announcement. This guarantees that the sensitive policy and key data for the group and potential rekey data for this individual cannot be read by anyone but the intended recipient.

If any error occurs during KeyDL message processing, regardless of whether the GM is in Terse or Verbose Mode, the registration session MUST be terminated, the GM MUST send a Key Download - Ack/Failure message, and all saved state information MUST be cleared. If in Terse Mode, the Notification Payload will be of type NACK to indicate termination. If in Verbose Mode, the Notification Payload will contain the type of error encountered.

5.2.1.3. Request to Join Error

The exchange type for Request to Join Error is eleven (11).

The components of the Request to Join Error Message are shown in Table 3:

Table 3: Request to Join Error (RTJ-Err) Message Definition

```
Message Name  : Request to Join Error (RTJ-Err)
Dissection    : {HDR-GrpID, [Nonce_I], Notification, [VendorID]}
Payload Types : GSAKMP Header, [Nonce] Notification, [Vendor ID]
```

In response to an unacceptable RTJ, the GC/KS MAY send a Request to Join Error (RTJ-Err) message containing an appropriate Notification payload. Note that the RTJ-Err message is not a signed message for the following reasons: the lack of awareness on the GM's perspective of who is a valid GC/KS as well as the need to protect the GC/KS from signing messages and using valuable resources. Following the sending of an RTJ-Err, the GC/KS MUST terminate the session, and all saved state information MUST be cleared.

Upon receipt of an RTJ-Err message, the GM will validate the following: the GroupID in the header belongs to a group to which the GM has sent an RTJ, and, if present, the Nonce_I matches a Nonce_I sent in an RTJ to that group. If the above checks are successful, the GM MAY terminate the state associated with that GroupID and

nonce. The GM SHOULD be capable of receiving a valid KeyDownload message for that GroupID and nonce after receiving an RTJ-Err for a locally configured amount of time.

5.2.1.4. Key Download - Ack/Failure

The exchange type for Key Download - Ack/Failure is four (4).

The components of the Key Download - Ack/Failure Message are shown in Table 4:

Table 4: Key Download - Ack/Failure (KeyDL-A/F) Message Definition

```
Message Name  : Key Download - Ack/Failure (KeyDL-A/F)
Dissection    : {HDR-GrpID, [Nonce_C], Notif_Ack, [VendorID]}SigM
Payload Types : GSAKMP Header, [Nonce], Notification, [Vendor
                ID], Signature
  SigM        : Signature of Group Member
  {}SigX      : Indicates fields used in Signature
```

In response to a properly processed KeyDL message, the GM creates and sends the KeyDL-A/F message. As defined in the dissection of the message, this message MUST contain payloads to hold the following information: Notification payload of type Acknowledgement (ACK) and signature information. If synchronized time is not available, the Nonce payload MUST be present for freshness, and the nonce value transmitted MUST be the GM's generated Nonce_C value. If the GM does not receive a KeyDL message within a locally configured amount of time, the GM MAY send a new RTJ. If the GM receives a valid LOA (see Section 5.2.1.5) message from the GC/KS before receipt of a KeyDL message, the GM SHOULD send a KeyDL-A/F message of type NACK followed by a new RTJ.

The GC/KS MUST be able to process the KeyDL-A/F message. <Process KeyDL-A/F> indicates the GC/KS actions that will determine how the KeyDL-A/F message will be acted upon. The following checks SHOULD be performed in the order presented.

In this procedure, the GC/KS will verify that the message header is properly formed and confirm that this message is for this group by checking the value of the GroupID. If the header checks pass, the GC/KS MUST check the message for freshness. If using nonces, the GC/KS MUST use its saved Nonce_C value and compare it for equality with the received Nonce_C value. If not using nonces, the GC/KS MUST check the timestamp in the Signature payload to determine if the message is new. After freshness is confirmed, the signature MUST be verified to ensure its authenticity. The GC/KS MUST use verified and trusted authentication material from a known root. If the message

signature verifies, the GC/KS processes the Notification payload. If the notification type is of type ACK, then the registration has completed successfully, and both parties SHOULD remove state information associated with this GM's registration.

If the GC/KS does not receive a KeyDL-A/F message of proper form or is unable to correctly process the KeyDL-A/F message, the Notification payload type is any value except ACK; or if no KeyDL-A/F message is received within the locally configured timeout, the GC/KS MUST evict this GM from the group in the next policy-defined Rekey Event. The GC/KS MAY send the OPTIONAL Lack_of_Ack message if running in Verbose Mode as defined in Section 5.2.1.5.

5.2.1.5. Lack of Ack

The exchange type for Lack of Ack is twelve (12).

The components of a Lack of Ack Message are shown in Table 5:

Table 5: Lack of Ack (LOA) Message Definition

```
Message Name  : Lack of Ack (LOA)
Dissection    : {HDR-GrpID, Member ID, [Nonce_R, Nonce_C],
                Notification, [VendorID]} SigC, [Cert]
Payload Types : GSAKMP Header, Identification, [Nonce],
                Notification, [Vendor ID], Signature,
                [Certificate]

  SigC        : Signature of Group Controller Key Server
  Cert        : Necessary Certificates, zero or more
  {}SigX      : Indicates fields used in Signature
  []          : Indicate an optional data item
```

If the GC/KS's local timeout value expires prior to receiving a KeyDL-A/F from the GM, the GC/KS MAY create and send a LOA message to the GM. As defined in the dissection of the message, this message MUST contain payloads to hold the following information: GM identification, Notification of error, and signature information.

If synchronized time is not available, the Nonce payloads MUST be present for freshness, and the nonce values transmitted MUST be the GC/KS's generated Nonce_R value and the combined Nonce_C value which was generated by using the GC/KS's Nonce_R value and the Nonce_I value received from the GM in the RTJ. These values were already generated during the Key Download message phase.

The GM MAY be able to process the LOA message based upon local configuration. <Process LOA> indicates the GM actions that will determine how the LOA message will be acted upon. The following checks SHOULD be performed in the order presented.

In this procedure, the GM MUST verify that the message header is properly formed and confirm that this message is for this group by checking the value of the GroupID. If the header checks pass, the GM MUST confirm that this message was intended for itself by comparing the Member ID in the Identification payload to its identity. After identification confirmation, the freshness values are checked. If using nonces, the GM MUST use its save Nonce_I value, extract the received GC/KS Nonce_R value, compute the combined Nonce_C value, and compare it to the received Nonce_C value. If not using nonces, the GM MUST check the timestamp in the Signature payload to determine if the message is new. After freshness is confirmed, access control checks MUST be performed on the GC/KS to determine its authority within this group. Then signature MUST be verified to ensure its authenticity, The GM MUST use verified and trusted authentication material from a known root.

If the checks succeed, the GM SHOULD resend a KeyDL-A/F for that session.

5.2.2. Cookies: Group Establishment with Denial of Service Protection

This section defines an OPTIONAL capability that MAY be implemented into GSAKMP when using IP-based groups. The information in this section borrows heavily from [IKEv2] as this protocol has already worked through this issue and GSAKMP is copying this concept. This section will contain paraphrased sections of [IKEv2] modified for GSAKMP to define the purpose of Cookies.

An optional Cookie mode is being defined for the GSAKMP to help against DoS attacks.

The term "cookies" originates with Karn and Simpson [RFC2522] in Photuris, an early proposal for key management with IPSec. The ISAKMP fixed message header includes two eight-octet fields titled "cookies". Instead of placing this cookie data in the header, in GSAKMP this data is moved into a Notification payload.

An expected attack against GSAKMP is state and CPU exhaustion, where the target GC/KS is flooded with Request to Join requests from forged IP addresses. This attack can be made less effective if a GC/KS implementation uses minimal CPU and commits no state to the communication until it knows the initiator potential GM can receive packets at the address from which it claims to be sending them. To

accomplish this, the GC/KS (when operating in Cookie mode) SHOULD reject initial Request to Join messages unless they contain a Notification payload of type "cookie". It SHOULD instead send a Cookie Download message as a response to the RTJ and include a cookie in a notify payload of type Cookie_Required. Potential GMs who receive such responses MUST retry the Request to Join message with the responder-GC/KS-supplied cookie in its notification payload of type Cookie, as defined by the optional Notification payload of the Request to Join Msg in Section 5.2.1.1. This initial exchange will then be as shown in Figure 2 with the components of the new message Cookie Download shown in Table 6. The exchange type for Cookie Download is ten (10).

Figure 2: GSAKMP Ladder Diagram with Cookies

Table 6: Cookie Download Message Definition

Message Name	: Cookie Download
Dissection	: {HDR-GrpID, Notif_COOKIE_REQUIRED, [VendorID]}
Payload Types	: GSAKMP Header, Notification, [Vendor ID]

The first two messages do not affect any GM or GC/KS state except for communicating the cookie.

A GSAKMP implementation SHOULD implement its GC/KS cookie generation in such a way as not to require any saved state to recognize its valid cookie when the second Request to Join message arrives. The exact algorithms and syntax they use to generate cookies does not affect interoperability and hence is not specified here.

The following is an example of how an endpoint could use cookies to implement limited DoS protection.

A good way to do this is to set the cookie to be:

Cookie = <SecretVersionNumber> | Hash(Ni | IPi | <secret>)

where <secret> is a randomly generated secret known only to the responder GC/KS and periodically changed, Ni is the nonce value taken from the initiator potential GM, and IPi is the asserted IP address of the candidate GM. The IP address is either the IP header's source IP address or else the IP address contained in the optional Notification "IPvalue" payload (if it is present). <SecretVersionNumber> should be changed whenever <secret> is regenerated. The cookie can be recomputed when the "Request to Join with Cookie Info" arrives and compared to the cookie in the received message. If it matches, the responder GC/KS knows that all values have been computed since the last change to <secret> and that IPi MUST be the same as the source address it saw the first time. Incorporating Ni into the hash assures that an attacker who sees only the Cookie_Download message cannot successfully forge a "Request to Join with Cookie Info" message. This Ni value MUST be the same Ni value from the original "Request to Join" message for the calculation to be successful.

If a new value for <secret> is chosen while connections are in the process of being initialized, a "Request to Join with Cookie Info" might be returned with a <SecretVersionNumber> other than the current one. The responder GC/KS in that case MAY reject the message by sending another response with a new cookie, or it MAY keep the old value of <secret> around for a short time and accept cookies computed from either one. The responder GC/KS SHOULD NOT accept cookies indefinitely after <secret> is changed, since that would defeat part of the denial of service protection. The responder GC/KS SHOULD change the value of <secret> frequently, especially if under attack.

An alternative example for Cookie value generation in a NAT environment is to substitute the IPi value with the IPValue received in the Notification payload in the RTJ message. This scenario is indicated by the presence of the Notification payload of type IPValue. With this substitution, a calculation similar to that described above can be used.

5.2.3. Group Establishment for Receive-Only Members

This section describes an OPTIONAL capability that may be implemented in a structured system where the authorized (S-)GC/KS is known in advance through out-of-band means and where synchronized time is available.

Unlike Standard Group Establishment, in the Receive-Only system, the GMs and (S-)GC/KSes operate in Terse Mode and exchange one message only: the Key Download. Potential new GMs do not send an RTJ. (S-)GC/KSes do not expect Key Download - ACK/Failure messages and do not remove GMs for lack or receipt of the message.

Operation is as follows: upon notification via an authorized out-of-band event, the (S-)GC/KS forms and sends a Key Download message to the new member with the Nonce payloads ABSENT. The GM verifies

- the ID payload identifies that GM

- the timestamp in the message is fresh

- the message is signed by an authorized (S-)GC/KS

- the signature on the message verifies

When using a Diffie-Hellman Key Creation Type for receive-only members, a static-ephemeral model is assumed: the Key Creation payload in the Key Download message contains the (S-)GC/KS's public component. The member's public component is assumed to be obtained through secure out-of-band means.

5.3. Group Maintenance

The Group Maintenance phase includes member joins and leaves, group rekey activities, policy updates, and group destruction. These activities are presented in the following sections.

5.3.1. Group Management

5.3.1.1. Rekey Events

A Rekey Event is any action, including a compromise report or key expiration, that requires the creation of a new group key and/or rekey information.

Once an event has been identified (as defined in the group security policy token), the GC/KS MUST create and provide a signed message containing the GTPK and rekey information to the group.

Each GM who receives this message MUST verify the signature on the message to ensure its authenticity. If the message signature does not verify, the message MUST be discarded. Upon verification, the GM will find the appropriate rekey download packet and decrypt the information with a stored rekey key(s). If a new Policy Token is distributed with the message, it MUST be encrypted in the old GTPK.

The exchange type for Rekey Event is five (5).

The components of a Rekey Event message are shown in Table 7:

Table 7: Rekey Event Message Definition

```
Message Name  : Rekey Event
Dissection    : {HDR-GrpID, ([Policy Token])*, Rekey Array,
                [VendorID]}SigC, [Cert]
Payload Types : GSAKMP Header, [Policy Token], Rekey Event,
                [Vendor ID], Signature, [Certificate],

  SigC        : Signature of Group Controller Key Server
  Cert        : Necessary Certificates, zero or more
  {}SigX      : Indicates fields used in Signature
  (data)*     : Indicates encrypted information
  []          : Indicate an optional data item
```

5.3.1.2. Policy Updates

New policy tokens are sent via the Rekey Event message. These policy updates may be coupled with an existing rekey event or may be sent in a message with the Rekey Event Type of the Rekey Event Payload set to None(0) (see Section 7.5.1).

A policy token MUST NOT be processed if the processing of the Rekey Event message carrying it fails. Policy token processing is type dependent and is beyond the scope of this document.

5.3.1.3. Group Destruction

Group destruction is also accomplished via the Rekey Event message. In a Rekey Event message for group destruction, the Sequence ID is set to 0xFFFFFFFF. Upon receipt of this authenticated Rekey Event message, group components MUST terminate processing of information associated with the indicated group.

5.3.2. Leaving a Group

There are several conditions under which a member will leave a group: eviction, voluntary departure without notice, and voluntary departure with notice (de-registration). Each of these is discussed in this section.

5.3.2.1. Eviction

At some point in the group's lifetime, it may be desirable to evict one or more members from a group. From a key management viewpoint, this involves revoking access to the group's protected data by "disabling" the departing members' keys. This is accomplished with a Rekey Event, which is discussed in more detail in Section 5.3.1.1. If future access to the group is also to be denied, the members MUST be added to a denied access control list, and the policy token's authorization rules MUST be appropriately updated so that they will exclude the expelled GM(s). After receipt of a new PT, GMs SHOULD evaluate the trustworthiness of any recent application data originating from the expelled GM(s).

5.3.2.2. Voluntary Departure without Notice

If a member wishes to leave a group for which membership imposes no cost or responsibility to that member, then the member MAY merely delete local copies of group keys and cease group activities.

5.3.2.3. De-Registration

If the membership in the group does impose cost or responsibility to the departing member, then the member SHOULD de-register from the group when that member wishes to leave. De-registration consists of a three-message exchange between the GM and the member's GC/KS: the Request_to_Depart, Departure_Response, and the Departure_Ack. Compliant GSAKMP implementations for GMs SHOULD support the de-registration messages. Compliant GSAKMP implementations for GC/KSes MUST support the de-registration messages.

5.3.2.3.1. Request to Depart

The Exchange Type for a Request_to_Depart Message is thirteen (13). The components of a Request_to_Depart Message are shown in Table 8.

Any GM desiring to initiate the de-registration process MUST generate and send an RTD message to notify the GC/KS of its intent. As defined in the dissection of the RTD message, this message MUST contain payloads to hold the following information: the GC/KS identification and Notification of the desire to leave the group.

When synchronization time is not available to the system as defined by the Policy Token, a Nonce payload MUST be included for freshness, and the Nonce_I value MUST be saved for later use. This message MUST then be signed by the GM.

Table 8: Request_to_Depart (RTD) Message Definition

```
  Message Name  : Request_to_Depart (RTD)
  Dissection    : {HDR-GrpID, GC/KS_ID, [Nonce_I], Notif_Leave_Group,
                   [VendorID]} SigM, [Cert]
  Payload Types : GSAKMP Header, Identification, [Nonce],
                  Notification, [Vendor ID], Signature,
                  [Certificate]

    SigM        : Signature of Group Member
    Cert        : Necessary Certificates, zero or more
    {}SigX      : Indicates fields used in Signature
    []          : Indicate an optional data item
```

Upon receipt of the RTD message, the GC/KS MUST verify that the message header is properly formed and confirm that this message is for this group by checking the value of the GroupID. If the header checks pass, then the identifier value in Identification payload is compared to its own, the GC/KS's identity, to confirm that the GM intended to converse with this GC/KS, the GC/KS who registered this member into the group. Then the identity of the sender is extracted from the Signature payload. This identity MUST be used to confirm that this GM is a member of the group serviced by this GC/KS. Then the GC/KS will confirm from the Notification payload that the GM is requesting to leave the group. Then the GC/KS will verify the signature on the message to ensure its authenticity. The GC/KS MUST use verified and trusted authentication material from a known root. If all checks pass and the message is successfully processed, then the GC/KS MUST form a Departure_Response message as defined in Section 5.3.2.3.2.

If the processing of the message fails, the de-registration session MUST be terminated, and all state associated with this session is removed. If the GC/KS is operating in Terse Mode, then no error message is sent to the GM. If the GC/KS is operating in Verbose Mode, then the GC/KS sends a Departure_Response Message with a Notification Payload of type Request_to_Depart_Error.

5.3.2.3.2. Departure_Response

The Exchange Type for a Departure_Response Message is fourteen (14). The components of a Departure_Response Message are shown in Table 9.

In response to a properly formed and verified RTD message, the GC/KS MUST create and send the DR message. As defined in the dissection of the message, this message MUST contain payloads to hold the following information: GM identification, Notification for acceptance of departure, and signature information. If synchronization time is not available, the Nonce payloads MUST be included in the message for freshness.

Table 9: Departure_Response (DR) Message Definition

```
Message Name  : Departure_Response (DR)
Dissection    : {HDR-GrpID, Member_ID, [Nonce_R, Nonce_C],
                Notification, [VendorID]} SigC, [Cert]
Payload Types : GSAKMP Header, Identification, [Nonce],
                Notification, [Vendor ID], Signature,
                [Certificate]

  SigC        : Signature of Group Member
  Cert        : Necessary Certificates, zero or more
  {}SigX      : Indicates fields used in Signature
  []          : Indicate an optional data item
```

If present, the nonce values transmitted MUST be the GC/KS's generated Nonce_R value and the combined Nonce_C value that was generated by using the GC/KS's Nonce_R value and the Nonce_I value received from the GM in the RTD. This Nonce_C value MUST be saved relative to this departing GM's ID.

The GM MUST be able to process the Departure_Response message. The following checks SHOULD be performed in the order presented.

The GM MUST verify that the message header is properly formed and confirm that this message is for this group by checking the value of the GroupID. If the header checks pass, the GM MUST confirm that this message was intended for itself by comparing the Member ID in the Identification payload to its identity. After identification confirmation, the freshness values are checked. If using nonces, the GM MUST use its saved Nonce_I value, extract the received GC/KS Nonce_R value, compute the combined Nonce_C value, and compare it for equality with the received Nonce_C value. If not using nonces, the GM MUST check the timestamp in the signature payload to determine if the message is new. After freshness is confirmed, confirmation of the identity of the signer of the DR message is the GMs authorized

GC/KS is performed. Then, the signature MUST be verified to ensure its authenticity. The GM MUST use verified and trusted authentication material from a known root. If the message signature verifies, then the GM MUST verify that the Notification is of Type Departure_Accepted or Request_to_Depart_Error.

If the processing is successful, and the Notification payload is of type Departure_Accepted, the member MUST form the Departure_ACK message as defined in Section 5.3.2.3.3. If the processing is successful, and the Notification payload is of type Request_to_Depart_Error, the member MUST remove all state associated with the de-registration session. If the member still desires to De-Register from the group, the member MUST restart the de-registration process.

If the processing of the message fails, the de-registration session MUST be terminated, and all state associated with this session is removed. If the GM is operating in Terse Mode, then a Departure_Ack Message with Notification Payload of type NACK is sent to the GC/KS. If the GM is operating in Verbose Mode, then the GM sends a Departure_Ack Message with a Notification Payload of the appropriate failure type.

5.3.2.3.3. Departure_ACK

The Exchange Type for a Departure_ACK Message is fifteen (15). The components of the Departure_ACK Message are shown in Table 10:

Table 10: Departure_ACK (DA) Message Definition

Message Name	: Departure_ACK (DA)
Dissection	: {HDR-GrpID, [Nonce_C], Notif_Ack, [VendorID]}SigM
Payload Types	: GSAKMP Header, [Nonce], Notification, [Vendor ID], Signature
SigM	: Signature of Group Member
{}SigX	: Indicates fields used in Signature

In response to a properly processed Departure_Response message, the GM MUST create and send the Departure_ACK message. As defined in the dissection of the message, this message MUST contain payloads to hold the following information: Notification payload of type Acknowledgement (ACK) and signature information. If synchronization time is not available, the Nonce payload MUST be present for freshness, and the nonce value transmitted MUST be the GM's generated Nonce_C value.

Upon receipt of the Departure_ACK, the GC/KS MUST perform the following checks. These checks SHOULD be performed in the order presented.

In this procedure, the GC/KS MUST verify that the message header is properly formed and confirm that this message is for this group by checking the value of the GroupID. If the header checks pass, the GC/KS MUST check the message for freshness. If using nonces, the GC/KS MUST use its saved Nonce_C value and compare it to the received Nonce_C value. If not using nonces, the GC/KS MUST check the timestamp in the signature payload to determine if the message is new. After freshness is confirmed, the signature MUST be verified to ensure its authenticity. The GC/KS MUST use verified and trusted authentication material from a known root. If the message signature verifies, the GC/KS processes the Notification payload. If the notification type is of type ACK, this is considered a successful processing of this message.

If the processing of the message is successful, the GC/KS MUST remove the member from the group. This MAY involve initiating a Rekey Event for the group.

If the processing of the message fails or if no Departure_Ack is received, the GC/KS MAY issue a LOA message.

6. Security Suite

The Security Definition Suite 1 MUST be supported. Other security suite definitions MAY be defined in other Internet specifications.

6.1. Assumptions

All potential GMs will have enough information available to them to use the correct Security Suite to join the group. This can be accomplished by a well-known default suite, 'Security Suite 1', or by announcing/posting another suite.

6.2. Definition Suite 1

GSAKMP implementations MUST support the following suite of algorithms and configurations. The following definition of Suite 1 borrows heavily from IKE's Oakley group 2 definition and Oakley itself.

The GSAKMP Suite 1 definition gives all the algorithm and cryptographic definitions required to process group establishment messages. It is important to note that GSAKMP does not negotiate

these cryptographic mechanisms. This definition is set by the Group Owner via the Policy Token (passed during the GSAKMP exchange for member verification purposes).

The GSAKMP Suite 1 definition is:

```
  Key download and Policy Token encryption algorithm definition:
  Algorithm:  AES
  Mode:       CBC
  Key Length: 128 bits

  Policy Token digital signature algorithm is:
    DSS-ASN1-DER
    Hash algorithm is:
    SHA-1

  Nonce Hash algorithm is:
    SHA-1

  The Key Creation definition is:
  Algorithm type is Diffie Hellman
  MODP group definition
  g:   2
  p:   "FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1"
       "29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD"
       "EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245"
       "E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED"
       "EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE65381"
       "FFFFFFFF FFFFFFFF"

  NOTE: The p and g values come from IKE [RFC2409], Section 6.2,
        "Second Oakley Group", and p is 1024 bits long.


  The GSAKMP message digital signature algorithm is:
  DSS-SHA1-ASN1-DER

  The digital signature ID type is:
  ID-DN-STRING
```

7. GSAKMP Payload Structure

A GSAKMP Message is composed of a GSAKMP Header (Section 7.1) followed by at least one GSAKMP Payload. All GSAKMP Payloads are composed of the Generic Payload Header (Section 7.2) followed by the specific payload data. The message is chained by a preceding payload defining its succeeding payload. Payloads are not required to be in the exact order shown in the message dissection in Section 5, provided that all required payloads are present. Unless it is explicitly stated in a dissection that multiple payloads of a single type may be present, no more than one payload of each type allowed by the message may appear. The final payload in a message will point to no succeeding payload.

All fields of type integer in the Header and Payload structure that are larger than one octet MUST be converted into Network Byte Order prior to data transmission.

Padding of fields MUST NOT be done as this leads to processing errors.

When a message contains a Vendor ID payload, the processing of the payloads of that message is modified as defined in Section 7.10.

7.1. GSAKMP Header

7.1.1. GSAKMP Header Structure

The GSAKMP Header fields are shown in Figure 3 and defined as:

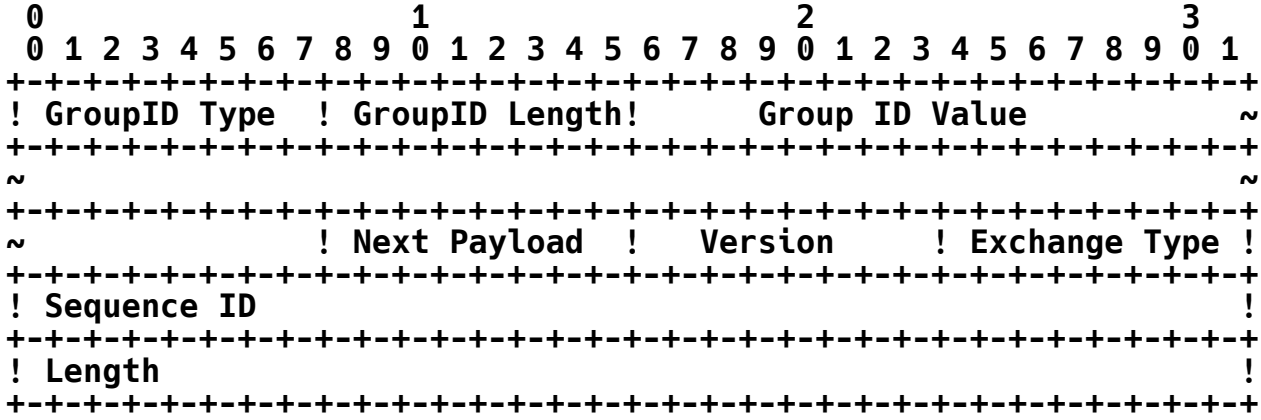

Figure 3: GSAKMP Header Format

Group Identification Type (1 octet) - Table 11 presents the group identification types. This field is treated as an unsigned value.

Table 11: Group Identification Types

Grp ID Type	Value	Description
Reserved	0	
UTF-8	1	Format defined in Section 7.1.1.1.1.
Octet String	2	This type MUST be implemented. Format defined in Section 7.1.1.1.2.
IPv4	3	Format defined in Section 7.1.1.1.3.
IPv6	4	Format defined in Section 7.1.1.1.4.
Reserved to IANA	5 - 192	
Private Use	193 - 255	

Group Identification Length (1 octet) - Length of the Group Identification Value field in octets. This value MUST NOT be zero (0). This field is treated as an unsigned value.

Group Identification Value (variable length) - Indicates the name/title of the group. All GroupID types should provide unique naming across groups. GroupID types SHOULD provide this capability by including a random element generated by the creator (owner) of the group of at least eight (8) octets, providing extremely low probability of collision in group names. The GroupID value is static throughout the life of the group.

Next Payload (1 octet) - Indicates the type of the next payload in the message. The format for each payload is defined in the following sections. Table 12 presents the payload types. This field is treated as an unsigned value.

Table 12: Payload Types

Next_Payload_Type	Value
None	0
Policy Token	1
Key Download Packet	2
Rekey Event	3
Identification	4
Reserved	5
Certificate	6
Reserved	7
Signature	8
Notification	9
Vendor ID	10
Key Creation	11
Nonce	12
Reserved to IANA	13 - 192
Private Use	193 - 255

Version (1 octet) - Indicates the version of the GSAKMP protocol in use. The current value is one (1). This field is treated as an unsigned value.

Exchange Type (1 octet) - Indicates the type of exchange (also known as the message type). Table 13 presents the exchange type values. This field is treated as an unsigned value.

Table 13: Exchange Types

Exchange_Type	Value
Reserved	0 - 3
Key Download Ack/Failure	4
Rekey Event	5
Reserved	6 - 7
Request to Join	8
Key Download	9
Cookie Download	10
Request to Join Error	11
Lack of Ack	12
Request to Depart	13
Departure Response	14
Departure Ack	15
Reserved to IANA	16 - 192
Private Use	193 - 255

Sequence ID (4 octets) - The Sequence ID is used for replay protection of group management messages. If the message is not a group management message, this value MUST be set to zero (0). The first value used by a (S-)GC/KS MUST be one (1). For each distinct group management message that this (S-)GC/KS transmits, this value MUST be incremented by one (1). Receivers of this group management message MUST confirm that the value received is greater than the value of the sequence ID received with the last group management message from this (S-)GC/KS. Group Components (e.g., GMs, S-GC/KSes) MUST terminate processing upon receipt of an authenticated group management message containing a Sequence ID of 0xFFFFFFFF. This field is treated as an unsigned integer in network byte order.

Length (4 octets) - Length of total message (header + payloads) in octets. This field is treated as an unsigned integer in network byte order.

7.1.1.1. GroupID Structure

This section defines the formats for the defined GroupID types.

7.1.1.1.1. UTF-8

The format for type UTF-8 [RFC3629] is shown in Figure 4.

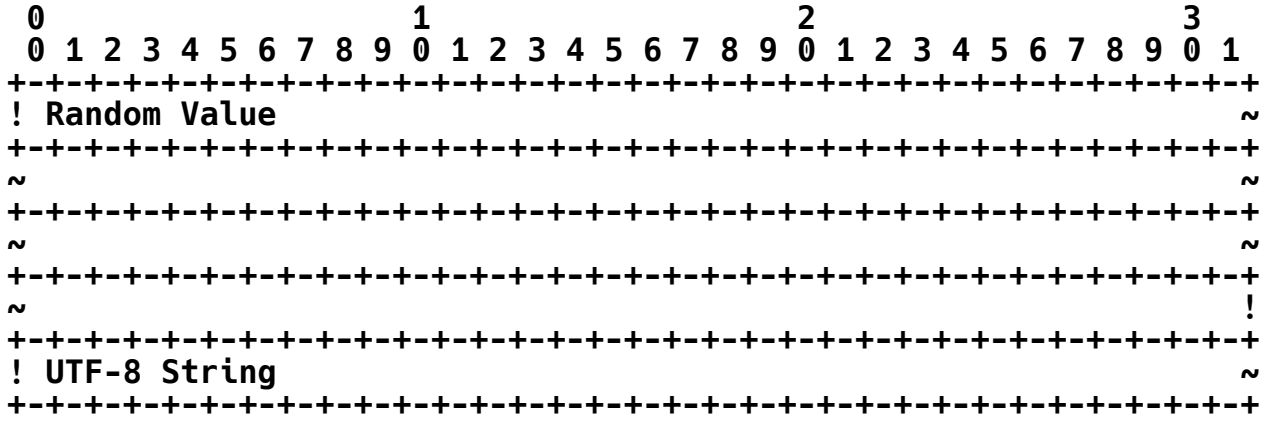

Figure 4: GroupID UTF-8 Format

Random Value (16 octets) - For the UTF-8 GroupID type, the Random Value is represented as a string of exactly 16 hexadecimal digits converted from its octet values in network-byte order. The leading zero hexadecimal digits and the trailing zero hexadecimal digits are always included in the string, rather than being truncated.

UTF-8 String (variable length) - This field contains the human readable portion of the GroupID in UTF-8 format. Its length is calculated as the (GroupID Length - 16) for the Random Value field. The minimum length for this field is one (1) octet.

7.1.1.1.2. Octet String

The format for type Octet String is shown in Figure 5.

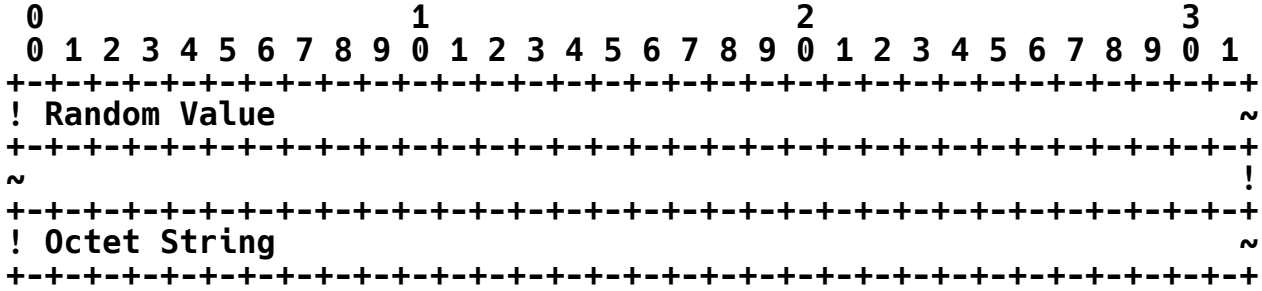

Figure 5: GroupID Octet String Format

Random Value (8 octets) - The 8-octet unsigned random value in network byte order format.

Octet String (variable length) - This field contains the Octet String portion of the GroupID. Its length is calculated as the (GroupID Length - 8) for the Random Value field. The minimum length for this field is one (1) octet.

7.1.1.1.3. IPv4 Group Identifier

The format for type IPv4 Group Identifier is shown in Figure 6.

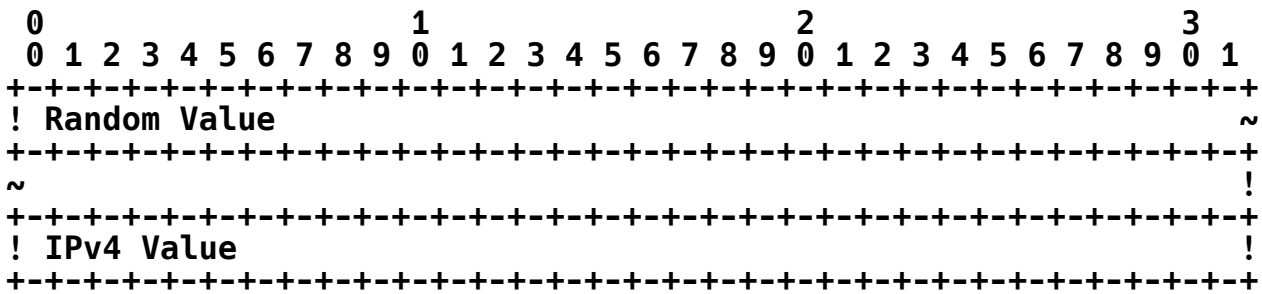

Figure 6: GroupID IPv4 Format

Random Value (8 octets) - The 8-octet unsigned random value in network byte order format.

IPv4 Value (4 octets) - The IPv4 value in network byte order format. This value MAY contain the multicast address of the group.

7.1.1.1.4. IPv6 Group Identifier

The format for type IPv6 Group Identifier is shown in Figure 7.

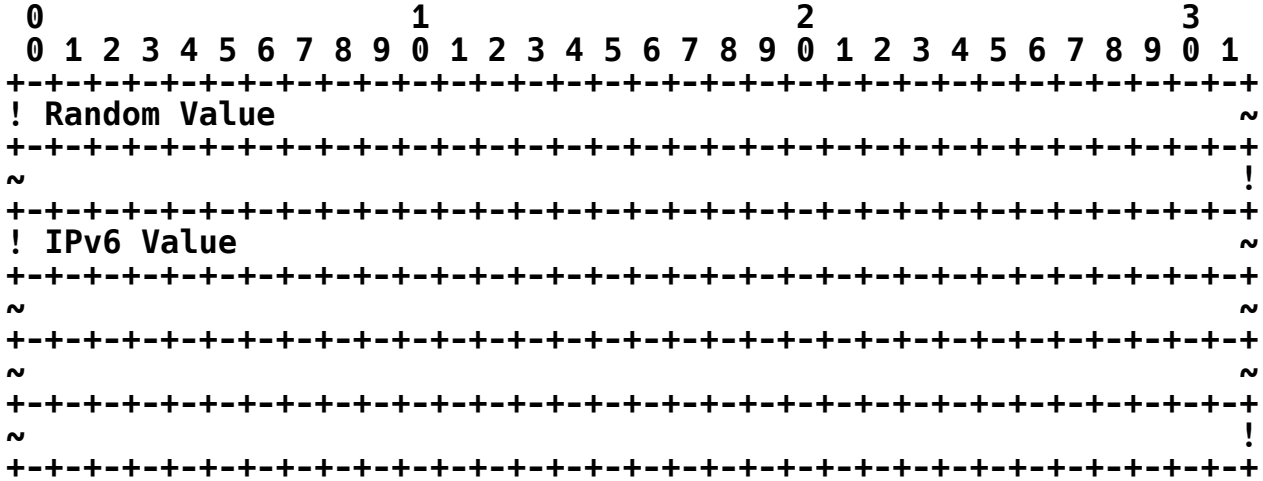

Figure 7: GroupID IPv6 Format

Random Value (8 octets) - The 8-octet unsigned random value in network byte order format.

IPv6 Value (16 octets) - The IPv6 value in network byte order format. This value MAY contain the multicast address of the group.

7.1.2. GSAKMP Header Processing

When processing the GSAKMP Header, the following fields MUST be checked for correct values:

1. Group ID Type - The Group ID Type value MUST be checked to be a valid group identification payload type as defined by Table 11. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

2. GroupID - The GroupID of the received message MUST be checked against the valid GroupIDs of the Group Component. If no match is found, then an error is logged; in addition, if in Verbose Mode, an appropriate message containing notification value Invalid-Group-ID will be sent.

3. Next Payload - The Next Payload value MUST be checked to be a valid payload type as defined by Table 12. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Invalid-Payload-Type will be sent.

4. Version - The GSAKMP version number MUST be checked that its value is one (1). For other values, see below for processing. The GSAKMP version number MUST be checked that it is consistent with the group's policy as specified in its Policy Token. If the version is not supported or authorized, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Invalid-Version will be sent.

5. Exchange Type - The Exchange Type MUST be checked to be a valid exchange type as defined by Table 13 and MUST be of the type expected to be received by the GSAKMP state machine. If the exchange type is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Invalid-Exchange-Type will be sent.

6. Sequence ID - The Sequence ID value MUST be checked for correctness. For negotiation messages, this value MUST be zero (0). For group management messages, this value MUST be greater than the last sequence ID received from this (S-)GC/KS. Receipt of incorrect Sequence ID on group management messages MUST NOT cause a reply message to be generated. Upon receipt of incorrect Sequence ID on non-group management messages, an error is logged. If in Verbose Mode, an appropriate message containing notification value Invalid-Sequence-ID will be sent.

The length fields in the GSAKMP Header (Group ID Length and Length) are used to help process the message. If any field is found to be incorrect, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

In order to allow a GSAKMP version one (v1) implementation to interoperate with future versions of the protocol, some ideas will be discussed here to this effect.

A (S-)GC/KS that is operating in a multi-versioned group as defined by the Policy Token can take many approaches on how to interact with the GMs in this group for a rekey message.

One possible solution is for the (S-)GC/KS to send out multiple rekey messages, one per version level that it supports. Then each GM would only process the message that has the version at which it is operating.

An alternative approach that all GM v1 implementations MUST support is the embedding of a v1 message inside a version two (v2) message. If a GM running at v1 receives a GSAKMP message that has a version value greater than one (1), the GM will attempt to process the information immediately after the Group Header as a Group Header for v1 of the protocol. If this is in fact a v1 Group Header, then the remainder of this v1 message will be processed in place. After processing this v1 embedded message, the data following the v1 message should be the payload as identified by the Next Payload field in the original header of the message and will be ignored by the v1 member. However, if the payload following the initial header is not a v1 Group Header, then the GM will gracefully handle the unrecognized message.

7.2. Generic Payload Header

7.2.1. Generic Payload Header Structure

Each GSAKMP payload defined in the following sections begins with a generic header, shown in Figure 8, that provides a payload "chaining" capability and clearly defines the boundaries of a payload. The Generic Payload Header fields are defined as follows:

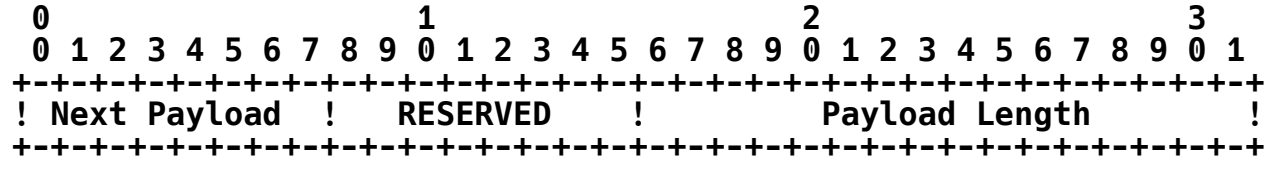

Figure 8: Generic Payload Header

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

7.2.2. Generic Payload Header Processing

When processing the Generic Payload Header, the following fields MUST be checked for correct values:

1. Next Payload - The Next Payload value MUST be checked to be a valid payload type as defined by Table 12. If the payload type is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Invalid-Payload-Type will be sent.

2. RESERVED - This field MUST contain the value zero (0). If the value of this field is not zero (0), then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

The length field in the Generic Payload Header is used to process the remainder of the payload. If this field is found to be incorrect, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

7.3. Policy Token Payload

7.3.1. Policy Token Payload Structure

The Policy Token Payload contains authenticatable group-specific information that describes the group security-relevant behaviors, access control parameters, and security mechanisms. Figure 9 shows the format of the payload.

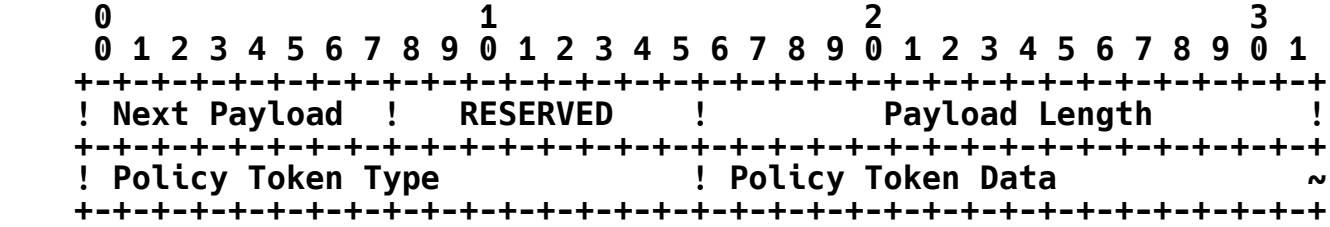

Figure 9: Policy Token Payload Format

The Policy Token Payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Policy Token Type (2 octets) - Specifies the type of Policy Token being used. Table 14 identifies the types of policy tokens. This field is treated as an unsigned integer in network byte order format.

Table 14: Policy Token Types

Policy_Token_Type	Value	Definition/Defined In
Reserved	0	
GSAKMP_ASN.1_PT_V1	1	All implementations of GSAKMP MUST support this PT format. Format specified in [RFC4534].
Reserved to IANA	2 - 49152	
Private Use	49153 - 65535	

Policy Token Data (variable length) - Contains Policy Token information. The values for this field are token specific, and the format is specified by the PT Type field.

If this payload is encrypted, only the Policy Token Data field is encrypted.

The payload type for the Policy Token Payload is one (1).

7.3.2. Policy Token Payload Processing

When processing the Policy Token Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. Policy Token Type - The Policy Token Type value MUST be checked to be a valid policy token type as defined by Table 14. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

3. Policy Token Data - This Policy Token Data MUST be processed according to the Policy Token Type specified. The type will define the format of the data.

7.4. Key Download Payload

Refer to the terminology section for the different terms relating to keys used within this section.

7.4.1. Key Download Payload Structure

The Key Download Payload contains group keys (e.g., group keys, initial rekey keys, etc.). These key download payloads can have several security attributes applied to them based upon the security policy of the group. Figure 10 shows the format of the payload.

The security policy of the group dictates that the key download payload MUST be encrypted with a key encryption key (KEK). The encryption mechanism used is specified in the Policy Token. The group members MUST create the KEK using the key creation method identified in the Key Creation Payload.

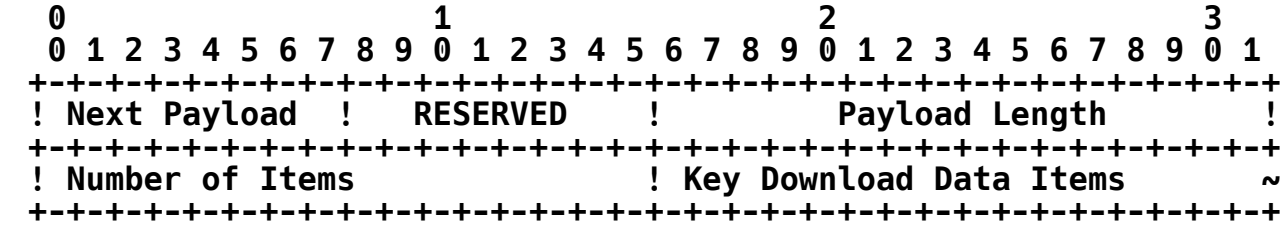

Figure 10: Key Download Payload Format

The Key Download Payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Number of Items (2 octets) - Contains the total number of group traffic protection keys and Rekey Arrays being passed in this data block. This field is treated as an unsigned integer in network byte order format.

Key Download Data Items (variable length) - Contains Key Download information. The Key Download Data is a sequence of Type/Length/Data of the Number of Items. The format for each item is defined in Figure 11.

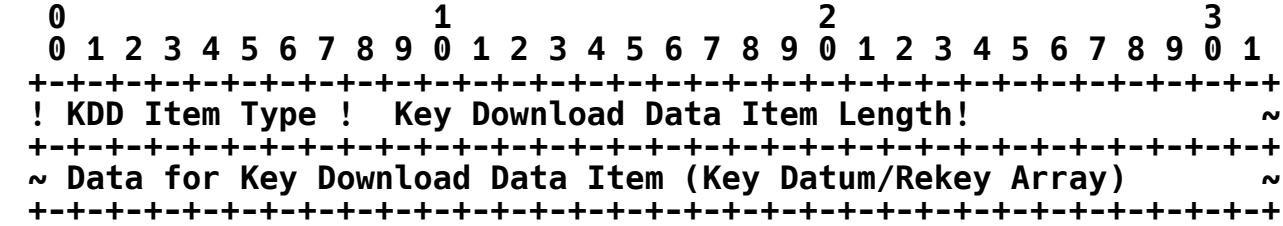

Figure 11: Key Download Data Item Format

For each Key Download Data Item, the data format is as follows:

Key Download Data (KDD) Item Type (1 octet) - Identifier for the type of data contained in this Key Download Data Item. See Table 15 for the possible values of this field. This field is treated as an unsigned value.

Key Download Data Item Length (2 octets) - Length in octets of the Data for the Key Download Data Item following this field. This field is treated as an unsigned integer in network byte order format.

Data for Key Download Data Item (variable length) - Contains Keys and related information. The format of this field is specific depending on the value of the Key Download Data Item Type field. For KDD Item Type of GTPK, this field will contain a Key Datum as defined in Section 7.4.1.1. For KDD Item Type Rekey - LKH, this field will contain a Rekey Array as defined in Section 7.4.1.2.

Table 15: Key Download Data Item Types

Key Download Data Item Type	Value	Definition
GTPK	0	This type MUST be implemented. This type identifies that the data contains group traffic protection key information.
Rekey - LKH	1	Optional
Reserved to IANA	2 - 192	
Private Use	193 - 255	

The encryption of this payload only covers the data subsequent to the Generic Payload header (Number of Items and Key Download Data Items fields).

The payload type for the Key Download Packet is two (2).

7.4.1.1. Key Datum Structure

A Key Datum contains all the information for a key. Figure 12 shows the format for this structure.

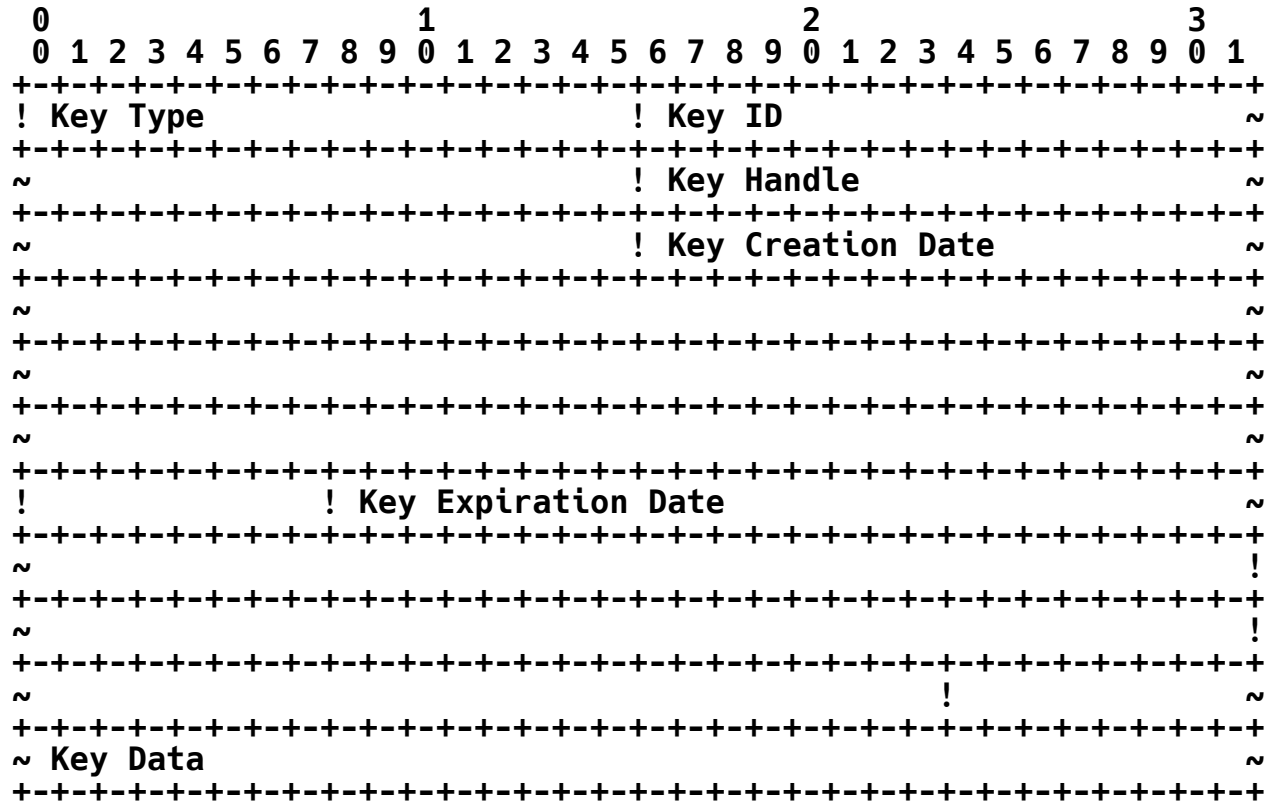

Figure 12: Key Datum Format

Key Type (2 octets) - This is the cryptographic algorithm for which this key data is to be used. This value is specified in the Policy Token. See Table 16 for the possible values of this field. This field is treated as an unsigned value.

Table 16: Cryptographic Key Types

Cryptographic_Key_Types	Value	Description/Defined In
Reserved	0 - 2	
3DES_CBC64_192	3	See [RFC2451].
Reserved	4 - 11	
AES_CBC_128	12	This type MUST be supported. See [IKEv2].
AES_CTR	13	See [IKEv2].
Reserved to IANA	14 - 49152	
Private Use	49153 - 65535	

Key ID (4 octets) - This is the permanent ID of all versions of the key. This value MAY be defined by the Policy Token. This field is treated as an octet string.

Key Handle (4 octets) - This is the value to uniquely identify a version (particular instance) of a key. This field is treated as an octet string.

Key Creation Date (15 octets) - This is the time value of when this key data was originally generated. This field contains the timestamp in UTF-8 format YYYYMMDDHHMMSSZ, where YYYY is the year (0000 - 9999), MM is the numerical value of the month (01 - 12), DD is the day of the month (01 - 31), HH is the hour of the day (00 - 23), MM is the minute within the hour (00 - 59), SS is the seconds within the minute (00 - 59), and the letter Z indicates that this is Zulu time. This format is loosely based on [RFC3161].

Key Expiration Date (15 octets) - This is the time value of when this key is no longer valid for use. This field contains the timestamp in UTF-8 format YYYYMMDDHHMMSSZ, where YYYY is the year (0000 - 9999), MM is the numerical value of the month (01 - 12), DD is the day of the month (01 - 31), HH is the hour of the day (00 - 23), MM is the minute within the hour (00 - 59), SS is the seconds within the minute (00 - 59), and the letter Z indicates that this is Zulu time. This format is loosely based on [RFC3161].

Key Data (variable length) - This is the actual key data, which is dependent on the Key Type algorithm for its format.

NOTE: The combination of the Key ID and the Key Handle MUST be unique within the group. This combination will be used to uniquely identify a key.

7.4.1.2. Rekey Array Structure

A Rekey Array contains the information for the set of KEKs that is associated with a Group Member. Figure 13 shows the format for this structure.

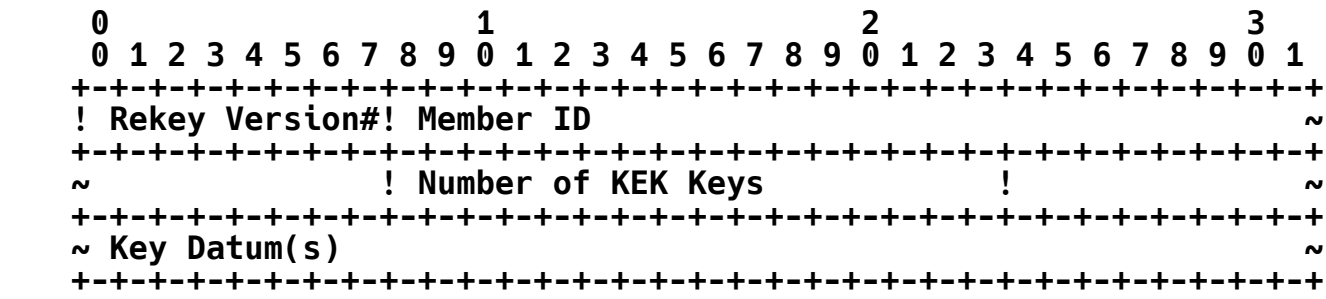

Figure 13: Rekey Array Structure Format

Rekey Version (1 octet) - Contains the version of the Rekey protocol in which the data is formatted. For Key Download Data Item Type of Rekey - LKH, refer to Section A.2 for a description of this value. This field is treated as an unsigned value.

Member ID (4 octets) - This is the Member ID of the Rekey sequence contained in this Rekey Array. This field is treated as an octet string. For Key Download Data Item Type of Rekey - LKH, refer to Section A.2 for a description of this value.

Number of KEK Keys (2 octets) - This value is the number of distinct KEK keys in this sequence. This value is treated as an unsigned integer in network byte order format.

Key Datum(s) (variable length) - The sequence of KEKs in Key Datum format. The format for each Key Datum in this sequence is defined in section 7.4.1.1.

Key ID (for Key ID within the Rekey) - LKH space, refer to Section A.2 for a description of this value.

7.4.2. Key Download Payload Processing

Prior to processing its data, the payload contents MUST be decrypted.

When processing the Key Download Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. KDD Item Type - All KDD Item Type fields MUST be checked to be a valid Key Download Data Item type as defined by Table 15. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

3. Key Type - All Key Type fields MUST be checked to be a valid encryption type as defined by Table 16. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Invalid-Key-Information will be sent.

4. Key Expiration Date - All Key Expiration Date fields MUST be checked confirm that their values represent a future and not a past time value. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Invalid-Key-Information will be sent.

The length and counter fields in the payload are used to help process the payload. If any field is found to be incorrect, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

7.5. Rekey Event Payload

Refer to the terminology section for the different terms relating to keys used within this section.

7.5.1. Rekey Event Payload Structure

The Rekey Event Payload MAY contain multiple keys encrypted in Wrapping KEKs. Figure 14 shows the format of the payload. If the data to be contained within a Rekey Event Payload is too large for the payload, the sequence can be split across multiple Rekey Event Payloads at a Rekey Event Data boundary.

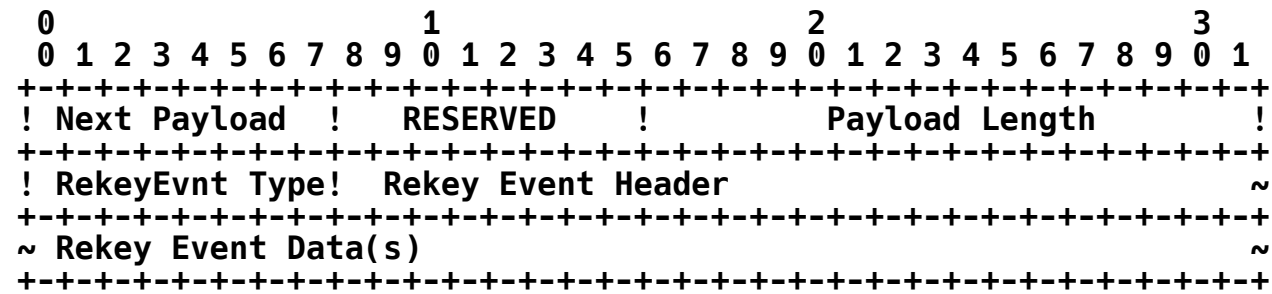

Figure 14: Rekey Event Payload Format

The Rekey Event Payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Rekey Event Type (1 octet) - Specifies the type of Rekey Event being used. Table 17 presents the types of Rekey events. This field is treated as an unsigned value.

Rekey Event Header (variable length) - This is the header information for the Rekey Event. The format for this is defined in Section 7.5.1.1, "Rekey Event Header Structure".

Rekey Event Data(s) (variable length) - This is the rekey information for the Rekey Event. The format for this is defined in Section 7.5.1.2, "Rekey Event Data Structure".

The Rekey Event payload type is three (3).

Table 17: Rekey Event Types

Rekey_Event_Type	Value	Definition/Defined In
None	0	This type MUST be implemented. In this case, the size of the Rekey Event Data field will be zero bytes long. The purpose of a Rekey Event Payload with type None is when it is necessary to send out a new token with no rekey information. GSAKMP rekey msg requires a Rekey Event Payload, and in this instance it would have rekey data of type None.
GSAKMP_LKH	1	The rekey data will be of type LKH formatted according to GSAKMP. The format for this field is defined in Section 7.5.1.2.
Reserved to IANA	2 - 192	
Private Use	193 - 255	

7.5.1.1. Rekey Event Header Structure

The format for the Rekey Event Header is shown in Figure 15.

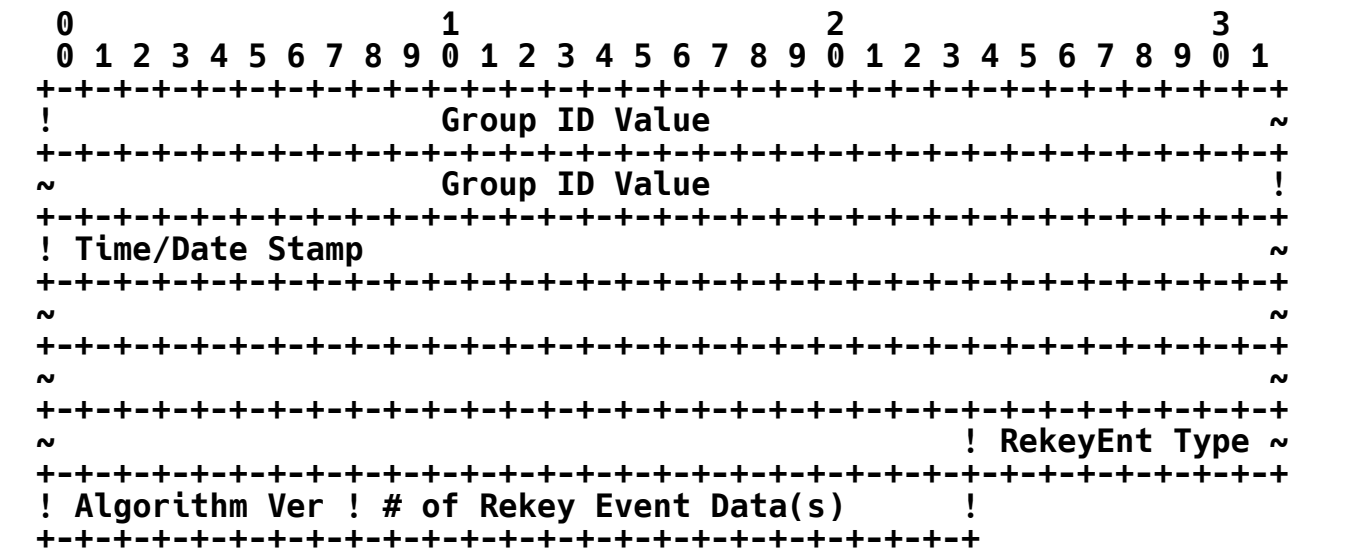

Figure 15: Rekey Event Header Format

Group Identification Value (variable length) - Indicates the name/title of the group to be rekeyed. This is the same format, length, and value as the Group Identification Value in Section 7.1, "GSAKMP Header".

Time/Date Stamp (15 octets) - This is the time value when the Rekey Event Data was generated. This field contains the timestamp in UTF-8 format YYYYMMDDHHMMSSZ, where YYYY is the year (0000 - 9999), MM is the numerical value of the month (01 - 12), DD is the day of the month (01 - 31), HH is the hour of the day (00 - 23), MM is the minute within the hour (00 - 59), SS is the seconds within the minute (00 - 59), and the letter Z indicates that this is Zulu time. This format is loosely based on [RFC3161].

Rekey Event Type (1 octet) - This is the Rekey algorithm being used for this group. The values for this field can be found in Table 17. This field is treated as an unsigned value.

Algorithm Version (1 octet) - Indicates the version of the Rekey Type being used. For Rekey Event Type of GSAKMP_LKH, refer to Section A.2 for a description of this value. This field is treated as an unsigned value.

# of Rekey Event Data(s) (2 octets) - The number of Rekey Event Data(s) contained in the Rekey Data. This value is treated as an unsigned integer in network byte order.

7.5.1.2. Rekey Event Data Structure

As defined in the Rekey Event Header, # of Rekey Data(s) field, multiple pieces of information are sent in a Rekey Event Data. Each end user, will be interested in only one Rekey Event Data among all of the information sent. Each Rekey Event Data will contain all the Key Packages that a user requires. For each Rekey Event Data, the data following the Wrapping fields is encrypted with the key identified in the Wrapping Header. Figure 16 shows the format of each Rekey Event Data.

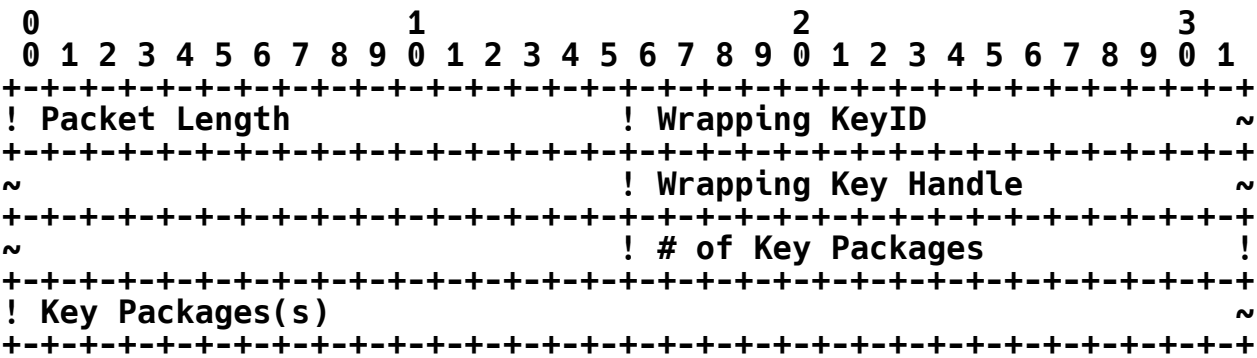

Figure 16: Rekey Event Data Format

Packet Length (2 octets) - Length in octets of the Rekey Event Data, which consists of the # of Key Packages and the Key Packages(s). This value is treated as an unsigned integer in network byte order.

Wrapping KeyID (4 octets) - This is the Key ID of the KEK that is being used for encryption/decryption of the new (rekeyed) keys. For Rekey Event Type of Rekey - LKH, refer to Section A.2 for a description of this value.

Wrapping Key Handle (4 octets) - This is a Key Handle of the KEK that is being used for encryption/decryption of the new (rekeyed) keys. Refer to Section 7.4.1.1 for the values of this field.

# of Key Packages (2 octets) - The number of key packages contained in this Rekey Event Data. This value is treated as an unsigned integer in network byte order.

Key Package(s) (variable length) - The type/length/value format of a Key Datum. The format for this is defined in Section 7.5.1.2.1.

7.5.1.2.1. Key Package Structure

Each Key Package contains all the information about the key. Figure 17 shows the format for a Key Package.

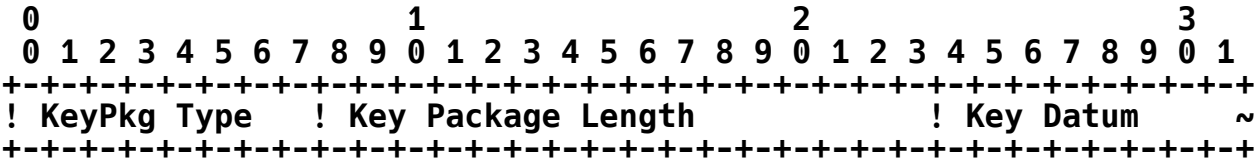

Figure 17: Key Package Format

Key Package Type (1 octet) - The type of key in this key package. Legal values for this field are defined in Table 15, Key Download Data Types. This field is treated as an unsigned value.

Key Package Length (2 octets) - The length of the Key Datum. This field is treated as an unsigned integer in network byte order format.

Key Datum (variable length) - The actual data of the key. The format for this field is defined in Section 7.4.1.1, "Key Datum Structure".

7.5.2. Rekey Event Payload Processing

When processing the Rekey Event Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. Rekey Event Type field within "Rekey Event" payload header - The Rekey Event Type MUST be checked to be a valid rekey event type as defined by Table 17. If the Rekey Event Type is not valid, then regardless of mode (e.g., Terse or Verbose) an error is logged. No response error message is generated for receipt of a Group Management Message.

3. Group ID Value - The Group ID value of the Rekey Event Header received message MUST be checked against the GroupID of the Group Component. If no match is found, the payload is discarded, then regardless of mode (e.g., Terse or Verbose) an error is logged. No response error message is generated for receipt of a Group Management Message.

4. Date/Time Stamp - The Date/Time Stamp value of the Rekey Event Header MAY be checked to determine if the Rekey Event generation time is recent relative to network delay and processing times. If the TimeStamp is judged not to be recent, an error is logged. No response error message is generated for receipt of a Group Management Message.

5. Rekey Event Type field within the "Rekey Event Header" - The Rekey Event Type of the Rekey Event Header received message MUST be checked to be a valid rekey event type, as defined by Table 17, and the same value of the Rekey Event Type earlier in this payload. If the Rekey Event Type is not valid or not equal to the previous value of the Rekey Event Type, then regardless of

mode (e.g., Terse or Verbose) an error is logged. No response error message is generated for receipt of a Group Management Message.

6. Algorithm Version - The Rekey Algorithm Version number MUST be checked to ensure that the version indicated is supported. If it is not supported, then regardless of mode (e.g., Terse or Verbose) an error is logged. No response error message is generated for receipt of a Group Management Message.

The length and counter fields are used to help process the message. If any field is found to be incorrect, then termination processing MUST be initiated.

A GM MUST process all the Rekey Event Datas as based on the rekey method used there is a potential that multiple Rekey Event Datas are for this GM. The Rekey Event Datas are processed in order until all Rekey Event Datas are consumed.

1. Wrapping KeyID - The Wrapping KeyID MUST be checked against the list of stored KEKs that this GM holds. If a match is found, then continue processing this Rekey Event Data. Otherwise, skip to the next Rekey Event Data.

2. Wrapping Handle - If a matching Wrapping KeyID was found, then the Wrapping Handle MUST be checked against the handle of the KEK for which the KeyID was a match. If the handles match, then the GM will process the Key Packages associated with this Rekey Event Data. Otherwise, skip to the next Rekey Event Data.

If a GM has found a matching Wrapping KeyID and Wrapping Handle, the GM decrypts the remaining data in this Rekey Event Data according to policy using the KEK defined by the Wrapping KeyID and Handle. After decrypting the data, the GM extracts the # of Key Packages field to help process the subsequent Key Packages. The Key Packages are processed as follows:

1. Key Package Type - The Key Package Type MUST be checked to be a valid key package type as defined by Table 15. If the Key Package Type is not valid, then regardless of mode (e.g., Terse or Verbose) an error is logged. No response error message is generated for receipt of a Group Management Message.

2. Key Package Length - The Key Package Length is used to process the subsequent Key Datum information.

3. Key Type - The Key Type MUST be checked to be a valid key type as defined by Table 16. If the Key Package Type is not valid, then regardless of mode (e.g., Terse or Verbose) an error is logged. No response error message is generated for receipt of a Group Management Message.

4. Key ID - The Key ID MUST be checked against the set of Key IDs that this user maintains for this Key Type. If no match is found, then regardless of mode (e.g., Terse or Verbose) an error is logged. No response error message is generated for receipt of a Group Management Message.

5. Key Handle - The Key Handle is extracted as is and is used to be the new Key Handle for the Key currently associated with the Key Package's Key ID.

6. Key Creation Date - The Key Creation Date MUST be checked that it is subsequent to the Key Creation Date for the currently held key. If this date is prior to the currently held key, then regardless of mode (e.g., Terse or Verbose) an error is logged. No response error message is generated for receipt of a Group Management Message.

7. Key Expiration Date - The Key Expiration Date MUST be checked that it is subsequent to the Key Creation Date just received and that the time rules conform with policy. If the expiration date is not subsequent to the creation date or does not conform with policy, then regardless of mode (e.g., Terse or Verbose) an error is logged. No response error message is generated for receipt of a Group Management Message.

8. Key Data - The Key Data is extracted based on the length information in the key package.

If there were no errors when processing the Key Package, the key represented by the KeyID will have all of its data updated based upon the received information.

7.6. Identification Payload

7.6.1. Identification Payload Structure

The Identification Payload contains entity-specific data used to exchange identification information. This information is used to verify the identities of members. Figure 18 shows the format of the Identification Payload.

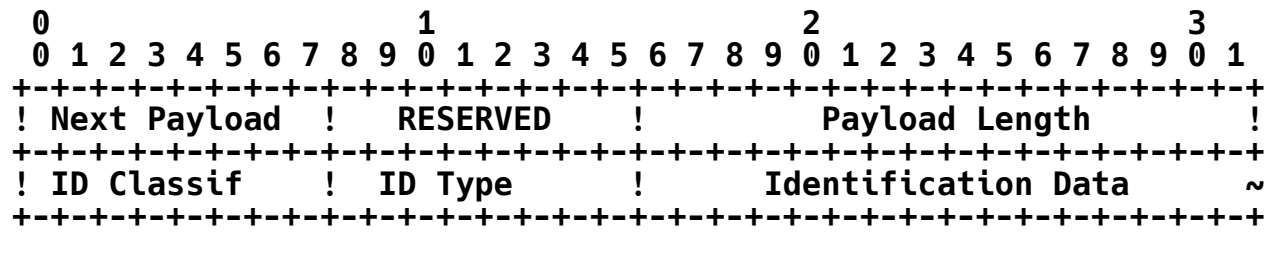

Figure 18: Identification Payload Format

The Identification Payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Identification (ID) Classification (1 octet) - Classifies the ownership of the Identification Data. Table 18 identifies possible values for this field. This field is treated as an unsigned value.

Table 18: Identification Classification

ID_Classification	Value
Sender	0
Receiver	1
Third Party	2
Reserved to IANA	3 - 192
Private Use	193 - 255

Identification (ID) Type (1 octet) - Specifies the type of Identification being used. Table 19 identifies possible values for this type. This field is treated as an unsigned value. All defined types are OPTIONAL unless otherwise stated.

Identification Data (variable length) - Contains identity information. The values for this field are group specific, and the format is specified by the ID Type field. The format for this field is stated in conjunction with the type in Table 19.

The payload type for the Identification Payload is four (4).

Table 19: Identification Types

ID_Type	Value	PKIX Cert Field	Description Defined In
Reserved	0		
ID_IPV4_ADDR	1	SubjAltName iPAddress	See [IKEv2] Section 3.5.
ID_FQDN	2	SubjAltName dNSName	See [IKEv2] Section 3.5.
ID_RFC822_ADDR	3	SubjAltName rfc822Name	See [IKEv2] Section 3.5.
Reserved	4		
ID_IPV6_ADDR	5	SubjAltName iPAddress	See [IKEv2] Section 3.5.
Reserved	6 - 8		
ID_DER_ASN1_DN	9	Entire Subject, bitwise Compare	See [IKEv2] Section 3.5.
Reserved	10		
ID_KEY_ID	11	N/A	See [IKEv2] Section 3.5.
Reserved	12 - 29		
Unencoded Name (ID_U_NAME)	30	Subject	The format for this type is defined in Section 7.6.1.1.
ID_DN_STRING	31	Subject	See [RFC4514]. This type MUST be implemented.
Reserved to IANA	32 - 192		
Private Use	193 - 255		

7.6.1.1. ID_U_NAME Structure

The format for type Unencoded Name (ID_U_NAME) is shown in Figure 19.

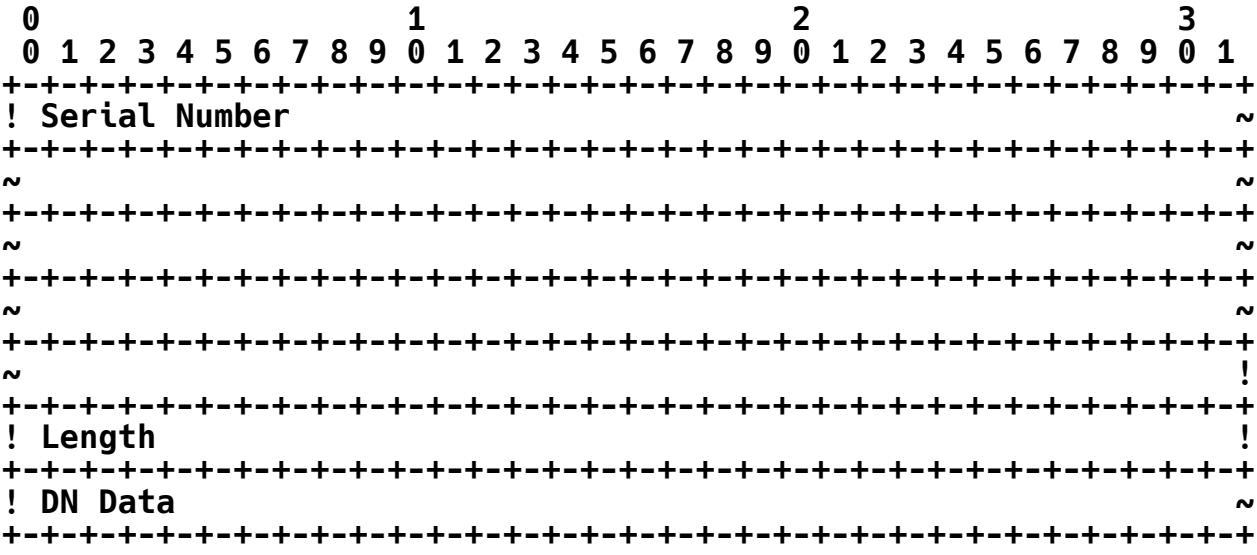

Figure 19: Unencoded Name (ID-U-NAME) Format

Serial Number (20 octets) - The certificate serial number. This field is treated as an unsigned integer in network byte order format.

Length (4 octets) - Length in octets of the DN Data field. This field is treated as an unsigned integer in network byte order format.

DN Data (variable length) - The actual UTF-8 DN value (Subject field) using the slash (/) character for field delimiters (e.g., "/C=US/ST=MD/L=Somewhere/O=ACME, Inc./OU=DIV1/CN=user1/ Email=user1@acme.com" without the surrounding quotes).

7.6.2. Identification Payload Processing

When processing the Identification Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. Identification Classification - The Identification Classification value MUST be checked to be a valid identification classification type as defined by Table 18. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

3. Identification Type - The Identification Type value MUST be checked to be a valid identification type as defined by Table 19. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

4. Identification Data - This Identification Data MUST be processed according to the identification type specified. The type will define the format of the data. If the identification data is being used to find a match and no match is found, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Invalid-ID-Information will be sent.

7.6.2.1. ID_U_NAME Processing

When processing the Identification Data of type ID_U_NAME, the following fields MUST be checked for correct values:

1. Serial Number - The serial number MUST be a greater than or equal to one (1) to be a valid serial number from a conforming CA [RFC3280]. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

2. DN Data - The DN data is processed as a UTF-8 string.

3. The CA MUST be a valid trusted policy creation authority as defined by the Policy Token.

These 2 pieces of information, Serial Number and DN Data, in conjunction, will then be used for party identification. These values are also used to help identify the certificate when necessary.

7.7. Certificate Payload

7.7.1. Certificate Payload Structure

The Certificate Payload provides a means to transport certificates or other certificate-related information via GSAKMP and can appear in any GSAKMP message. Certificate payloads SHOULD be included in an exchange whenever an appropriate directory service (e.g., LDAP [RFC4523]) is not available to distribute certificates. Multiple

certificate payloads MAY be sent to enable verification of certificate chains. Conversely, zero (0) certificate payloads may be sent, and the receiving GSAKMP MUST rely on some other mechanism to retrieve certificates for verification purposes. Figure 20 shows the format of the Certificate Payload.

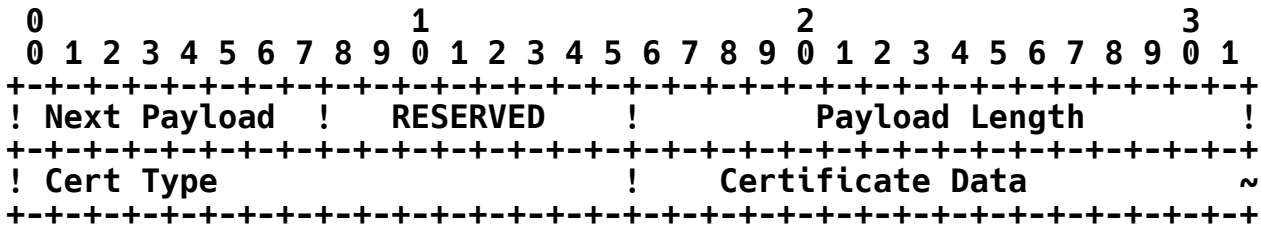

Figure 20: Certificate Payload Format

The Certificate Payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Certificate Type (2 octets) - This field indicates the type of certificate or certificate-related information contained in the Certificate Data field. Table 20 presents the types of certificate payloads. This field is treated as an unsigned integer in network byte order format.

Certificate Data (variable length) - Actual encoding of certificate data. The type of certificate is indicated by the Certificate Type/Encoding field.

The payload type for the Certificate Payload is six (6).

Table 20: Certificate Payload Types

Certificate_Type	Value	Description/ Defined In
None	0	
Reserved	1 - 3	
X.509v3 Certificate -- Signature -- DER Encoding	4	This type MUST be implemented. Contains a DER encoded X.509 certificate.
Reserved	5 - 6	
Certificate Revocation List (CRL)	7	Contains a BER encoded X.509 CRL.
Reserved	8 - 9	
X.509 Certificate -- Attribute	10	See [IKEv2], Sec 3.6.
Raw RSA Key	11	See [IKEv2], Sec 3.6.
Hash and URL of X.509 Certificate	12	See [IKEv2], Sec 3.6.
Hash and URL of X.509 bundle	13	See [IKEv2], Sec 3.6.
Reserved to IANA	14 -- 49152	
Private Use	49153 -- 65535	

7.7.2. Certificate Payload Processing

When processing the Certificate Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. Certificate Type - The Certificate Type value MUST be checked to be a valid certificate type as defined by Table 20. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Cert-Type-Unsupported will be sent.

3. Certificate Data - This Certificate Data MUST be processed according to the certificate type specified. The type will define the format of the data. Receipt of a certificate of the trusted policy creation authority in a Certificate payload causes

the payload to be discarded. This received certificate MUST NOT be used to verify the message. The certificate of the trusted policy creation authority MUST be retrieved by other means.

7.8. Signature Payload

7.8.1. Signature Payload Structure

The Signature Payload contains data generated by the digital signature function. The digital signature, as defined by the dissection of each message, covers the message from the GSAKMP Message Header through the Signature Payload, up to but not including the Signature Data Length. Figure 21 shows the format of the Signature Payload.

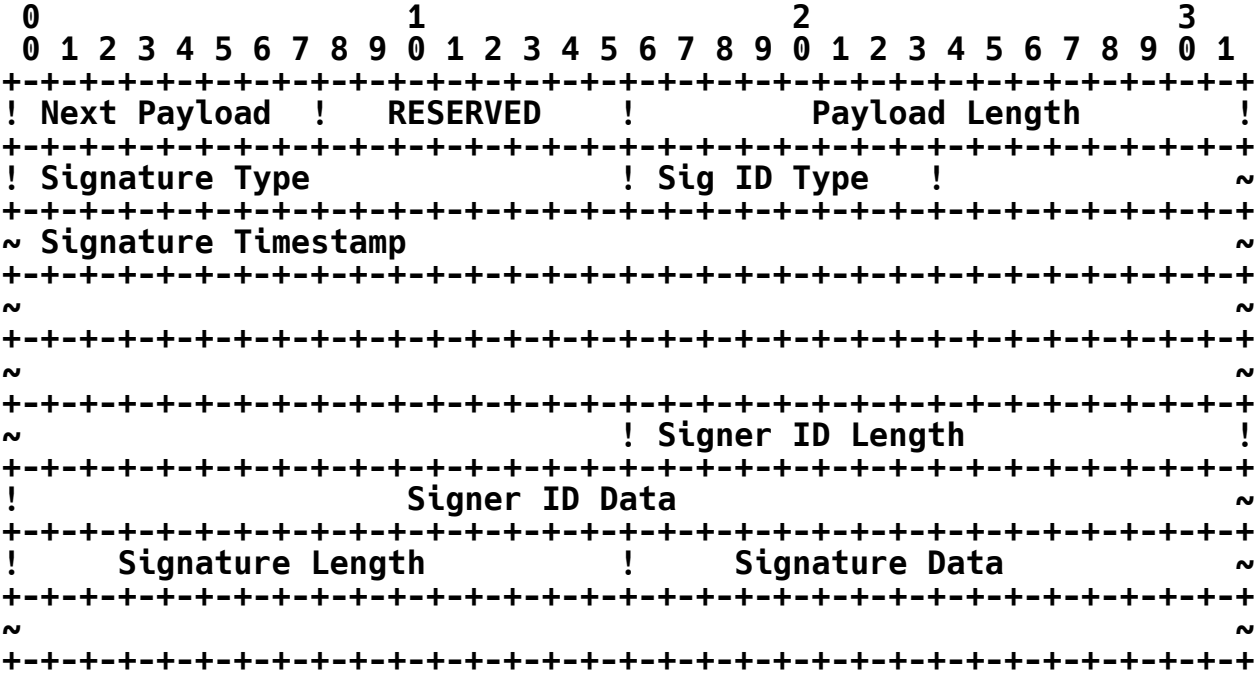

Figure 21: Signature Payload Format

The Signature Payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Signature Type (2 octets) - Indicates the type of signature. Table 21 presents the allowable signature types. This field is treated as an unsigned integer in network byte order format.

Table 21: Signature Types

Signature Type	Value	Description/ Defined In
DSS/SHA1 with ASN.1/DER encoding (DSS-SHA1-ASN1-DER)	0	This type MUST be supported.
RSA1024-MD5	1	See [RFC3447].
ECDSA-P384-SHA3	2	See [FIPS186-2].
Reserved to IANA	3 - 41952	
Private Use	41953 - 65536	

Signature ID Type (1 octet) - Indicates the format for the Signature ID Data. These values are the same as those defined for the Identification Payload Identification types, which can be found in Table 19. This field is treated as an unsigned value.

Signature Timestamp (15 octets) - This is the time value when the digital signature was applied. This field contains the timestamp in UTF-8 format YYYYMMDDHHMMSSZ, where YYYY is the year (0000 - 9999), MM is the numerical value of the month (01 - 12), DD is the day of the month (01 - 31), HH is the hour of the day (00 - 23), MM is the minute within the hour (00 - 59), SS is the seconds within the minute (00 - 59), and the letter Z indicates that this is Zulu time. This format is loosely based on [RFC3161].

Signer ID Length (2 octets) - Length in octets of the Signer's ID. This field is treated as an unsigned integer in network byte order format.

Signer ID Data (variable length) - Data identifying the Signer's ID (e.g., DN). The format for this field is based on the Signature ID Type field and is shown where that type is defined. The contents of this field MUST be checked against the Policy Token to determine the authority and access of the Signer within the context of the group.

Signature Length (2 octets) - Length in octets of the Signature Data. This field is treated as an unsigned integer in network byte order format.

Signature Data (variable length) - Data that results from applying the digital signature function to the GSAKMP message and/or payload.

The payload type for the Signature Payload is eight (8).

7.8.2. Signature Payload Processing

When processing the Signature Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. Signature Type - The Signature Type value MUST be checked to be a valid signature type as defined by Table 21. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

3. Signature ID Type - The Signature ID Type value MUST be checked to be a valid signature ID type as defined by Table 19. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

4. Signature Timestamp - This field MAY be checked to determine if the transaction signing time is fresh relative to expected network delays. Such a check is appropriate for systems in which archived sequences of events are desired.

   NOTE: The maximum acceptable age of a signature timestamp relative to the local system clock is a locally configured parameter that can be tuned by its GSAKMP management interface.

5. Signature ID Data - This field will be used to identify the sending party. This information MUST then be used to confirm that the correct party sent this information. This field is also used to retrieve the appropriate public key of the certificate to verify the message.

6. Signature Data - This value MUST be compared to the recomputed signature to verify the message. Information on how to verify certificates used to ascertain the validity of the signature can be found in [RFC3280]. Only after the certificate identified by the Signature ID Data is verified can the signature be computed to compare to the signature data for signature verification. A potential error that can occur during signature verification is Authentication-Failed. Potential errors that can occur while processing certificates for signature verification are: Invalid-Certificate, Invalid-Cert-Authority, Cert-Type-Unsupported, and Certificate-Unavailable.

The length fields in the Signature Payload are used to process the remainder of the payload. If any field is found to be incorrect, then termination processing MUST be initiated.

7.9. Notification Payload

7.9.1. Notification Payload Structure

The Notification Payload can contain both GSAKMP and group-specific data and is used to transmit informational data, such as error conditions, to a GSAKMP peer. It is possible to send multiple independent Notification payloads in a single GSAKMP message. Figure 22 shows the format of the Notification Payload.

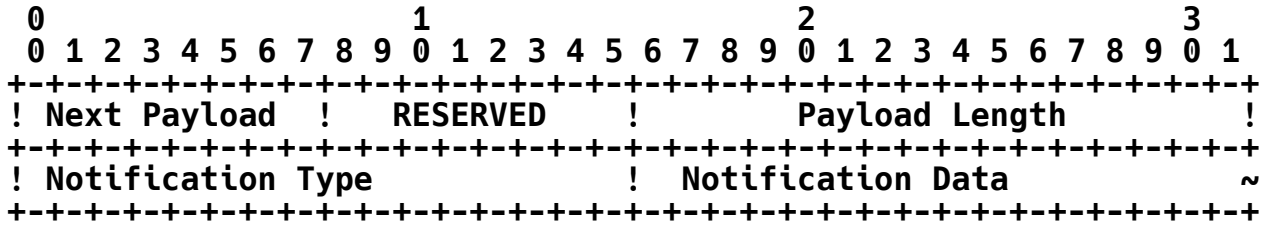

Figure 22: Notification Payload Format

The Notification Payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Notification Type (2 octets) - Specifies the type of notification message. Table 22 presents the Notify Payload Types. This field is treated as an unsigned integer in network byte order format.

Notification Data (variable length) - Informational or error data transmitted in addition to the Notify Payload Type. Values for this field are Domain of Interpretation (DOI) specific.

The payload type for the Notification Payload is nine (9).

Table 22: Notification Types

Notification Type	Value
None	0
Invalid-Payload-Type	1
Reserved	2 - 3
Invalid-Version	4
Invalid-Group-ID	5
Invalid-Sequence-ID	6
Payload-Malformed	7
Invalid-Key-Information	8
Invalid-ID-Information	9
Reserved	10 - 11
Cert-Type-Unsupported	12
Invalid-Cert-Authority	13
Authentication-Failed	14
Reserved	15 - 16
Certificate-Unavailable	17
Reserved	18
Unauthorized-Request	19
Reserved	20 - 22
Acknowledgement	23
Reserved	24 - 25
Nack	26
Cookie-Required	27
Cookie	28
Mechanism Choices	29
Leave Group	30
Departure Accepted	31
Request to Depart Error	32
Invalid Exchange Type	33
IPv4 Value	34

IPv6 Value	35
Prohibited by Group Policy	36
Prohibited by Locally Configured Policy	37
Reserved to IANA	38 - 49152
Private Use	49153 -- 65535

7.9.1.1. Notification Data - Acknowledgement (ACK) Payload Type

The data portion of the Notification payload of type ACK either serves as confirmation of correct receipt of the Key Download message or, when needed, provides other receipt information when included in a signed message. Figure 23 shows the format of the Notification Data - Acknowledge Payload Type.

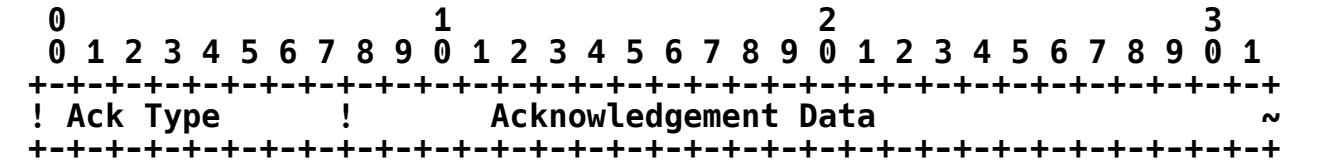

Figure 23: Notification Data - Acknowledge Payload Type Format

The Notification Data - Acknowledgement Payload Type data fields are defined as follows:

Ack Type (1 octet) - Specifies the type of acknowledgement. Table 23 presents the Notify Acknowledgement Payload Types. This field is treated as an unsigned value.

Table 23: Acknowledgement Types

ACK_Type	Value	Definition
Simple	0	Data portion null.
Reserved to IANA	1 - 192	
Private Use	193 - 255	

7.9.1.2. Notification Data - Cookie_Required and Cookie Payload Type

The data portion of the Notification payload of types Cookie_Required and Cookie contain the Cookie value. The value for this field will have been computed by the responder GC/KS and sent to the GM. The GM will take the value received and copy it into the Notification payload Notification Data field of type Cookie that is transmitted in the "Request to Join with Cookie Info" back to the GC/KS. The cookie value MUST NOT be modified.

The format for this is already described in the discussion on cookies in Section 5.2.2.

7.9.1.3. Notification Data - Mechanism Choices Payload Type

The data portion of the Notification payload of type Mechanism Choices contains the mechanisms the GM is requesting to use for the negotiation with the GC/KS. This information will be supplied by the GM in a RTJ message. Figure 24 shows the format of the Notification Data - Mechanism Choices Payload Type. Multiple type|length|data choices are strung together in one notification payload to allow a user to transmit all relevant information within one Notification Payload. The length of the payload will control the parsing of the Notification Data Mechanism Choices field.

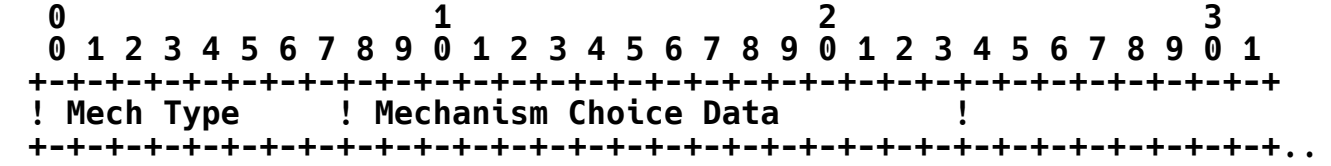

Figure 24: Notification Data - Mechanism Choices Payload Type Format

The Notification Data - Mechanism Choices Payload Type data fields are defined as follows:

Mechanism Type (1 octet) - Specifies the type of mechanism. Table 24 presents the Notify Mechanism Choices Mechanism Types. This field is treated as an unsigned value.

Table 24: Mechanism Types

Mechanism_Type	Value	Mechanism Choice Data Value Table Reference
Key Creation Algorithm	0	Table 26
Encryption Algorithm	1	Table 16
Nonce Hash Algorithm	2	Table 25
Reserved to IANA	3 - 192	
Private Use	193 - 255	

Mechanism Choice Data (2 octets) - The data value for the mechanism type being selected. The values are specific to each Mechanism Type defined. All tables necessary to define the values that are not defined elsewhere (in this specification or others) are defined here. This field is treated as an unsigned integer in network byte order format.

Table 25: Nonce Hash Types

Nonce_Hash_Type	Value	Description
Reserved	0	
SHA-1	1	This type MUST be supported.
Reserved to IANA	2 - 49152	
Private Use	49153 - 65535	

7.9.1.4. Notification Data - IPv4 and IPv6 Value Payload Types

The data portion of the Notification payload of type IPv4 and IPv6 value contains the appropriate IP value in network byte order. This value will be set by the creator of the message for consumption by the receiver of the message.

7.9.2. Notification Payload Processing

When processing the Notification Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. Notification Type - The Notification type value MUST be checked to be a notification type as defined by Table 22. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

3. Notification Data - This Notification Data MUST be processed according to the notification type specified. The type will define the format of the data. When processing this data, any type field MUST be checked against the appropriate table for correct values. If the contents of the Notification Data are not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

7.10. Vendor ID Payload

7.10.1. Vendor ID Payload Structure

The Vendor ID Payload contains a vendor-defined constant. The constant is used by vendors to identify and recognize remote instances of their implementations. This mechanism allows a vendor to experiment with new features while maintaining backwards compatibility. Figure 25 shows the format of the payload.

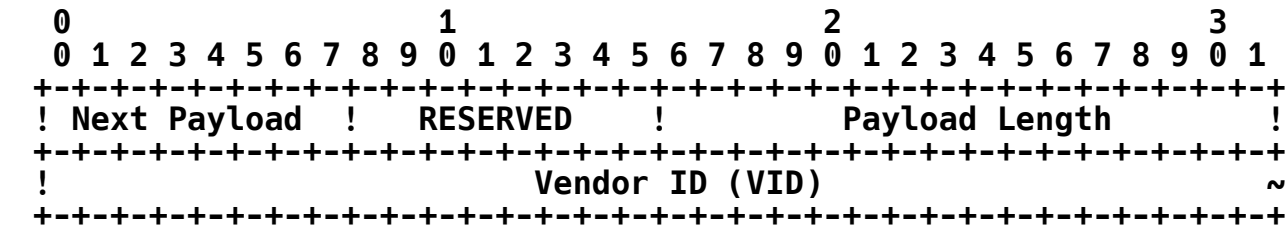

Figure 25: Vendor ID Payload Format

A Vendor ID payload MAY announce that the sender is capable of accepting certain extensions to the protocol, or it MAY simply identify the implementation as an aid in debugging. A Vendor ID payload MUST NOT change the interpretation of any information defined in this specification. Multiple Vendor ID payloads MAY be sent. An implementation is NOT REQUIRED to send any Vendor ID payload at all.

A Vendor ID payload may be sent as part of any message. Receipt of a familiar Vendor ID payload allows an implementation to make use of Private Use numbers described throughout this specification -- private payloads, private exchanges, private notifications, etc. This implies that all the processing rules defined for all the payloads are now modified to recognize all values defined by this Vendor ID for all fields of all payloads. Unfamiliar Vendor IDs MUST be ignored.

Writers of Internet-Drafts who wish to extend this protocol MUST define a Vendor ID payload to announce the ability to implement the extension in the Internet-Draft. It is expected that Internet-Drafts that gain acceptance and are standardized will be given assigned values out of the Reserved to IANA range, and the requirement to use a Vendor ID payload will go away.

The Vendor ID payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Vendor ID (variable length) - The Vendor ID value. The minimum length for this field is four (4) octets. It is the responsibility of the person choosing the Vendor ID to assure its uniqueness in spite of the absence of any central registry for IDs. Good practice is to include a company name, a person name, or similar type data. A message digest of a long unique string is preferable to the long unique string itself.

The payload type for the Vendor ID Payload is ten (10).

7.10.2. Vendor ID Payload Processing

When processing the Vendor ID Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. Vendor ID - The Vendor ID Data MUST be processed to determine if the Vendor ID value is recognized by the implementation. If the Vendor ID value is not recognized, then regardless of mode (e.g., Terse or Verbose) this information is logged. Processing of the message MUST continue regardless of recognition of this value.

It is recommended that implementations that want to use Vendor-ID-specific information attempt to process the Vendor ID payloads of an incoming message prior to the remainder of the message processing. This will allow the implementation to recognize that when processing other payloads it can use the larger set of values for payload fields (Private Use values, etc.) as defined by the recognized Vendor IDs.

7.11. Key Creation Payload

7.11.1. Key Creation Payload Structure

The Key Creation Payload contains information used to create key encryption keys. The security attributes for this payload are provided in the Policy Token. Figure 26 shows the format of the payload.

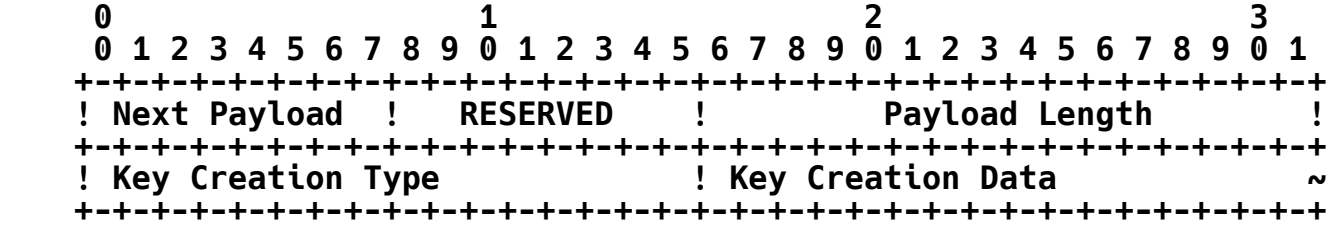

Figure 26: Key Creation Payload Format

The Key Creation Payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Key Creation Type (2 octets) - Specifies the type of Key Creation being used. Table 26 identifies the types of key creation information. This field is treated as an unsigned integer in network byte order format.

Key Creation Data (variable length) - Contains Key Creation information. The values for this field are group specific, and the format is specified by the key creation type field.

The payload type for the Key Creation Packet is eleven (11).

Table 26: Types of Key Creation Information

Key Creation Type	Value	Definition/Defined In
Reserved	0 - 1	
Diffie-Hellman 1024-bit MODP Group Truncated	2	This type MUST be supported. Defined in [IKEv2] B.2. If the output of the process is longer than needed for the defined mechanism, use the first X low order bits and truncate the remainder.
Reserved	3 - 13	
Diffie-Hellman 2048-bit MODP Group Truncated	14	Defined in [RFC3526]. If the output of the process is longer than needed for the defined mechanism, use the first X low order bits and truncate the remainder.
Reserved to IANA	15 - 49152	
Private Use	49153 - 65535	

7.11.2. Key Creation Payload Processing

The specifics of the Key Creation Payload are defined in Section 7.11.

When processing the Key Creation Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. Key Creation Type - The Key Creation Type value MUST be checked to be a valid key creation type as defined by Table 26. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

3. Key Creation Data - This Key Creation Data MUST be processed according to the key creation type specified to generate the KEK to protect the information to be sent in the appropriate message. The type will define the format of the data.

Implementations that want to derive other keys from the initial Key Creation keying material (for example, DH Secret keying material) MUST define a Key Creation Type other than one of those shown in Table 26. The new Key Creation Type must specify that derivation's algorithm, for which the KEK MAY be one of the keys derived.

7.12. Nonce Payload

7.12.1. Nonce Payload Structure

The Nonce Payload contains random data used to guarantee freshness during an exchange and protect against replay attacks. Figure 27 shows the format of the Nonce Payload.

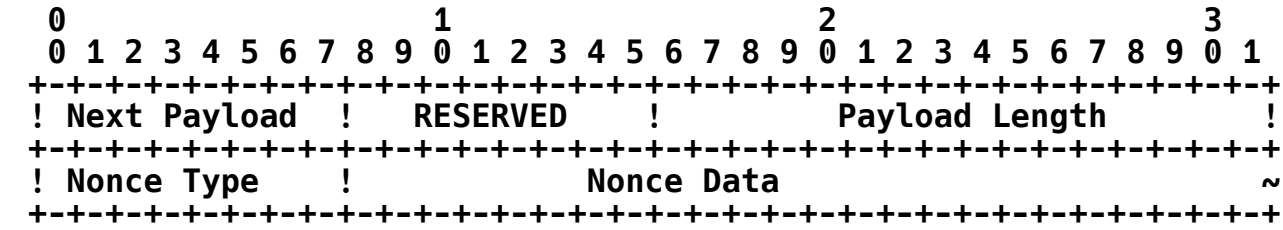

Figure 27: Nonce Payload Format

The Nonce Payload fields are defined as follows:

Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the "chaining" capability. Table 12 identifies the payload types. This field is treated as an unsigned value.

RESERVED (1 octet) - Unused, set to 0.

Payload Length (2 octets) - Length in octets of the current payload, including the generic payload header. This field is treated as an unsigned integer in network byte order format.

Nonce Type (1 octet) - Specifies the type of nonce being used. Table 27 identifies the types of nonces. This field is treated as an unsigned value.

Table 27: Nonce Types

Nonce_Type	Value	Definition
None	0	
Initiator (Nonce_I)	1	
Responder (Nonce_R)	2	
Combined (Nonce_C)	3	Hash (Append (Initiator_Value,Responder_Value)) The hash type comes from the Policy (e.g., Security Suite Definition of Policy Token).
Reserved to IANA	4 - 192	
Private Use	192 - 255	

Nonce Data (variable length) - Contains the nonce information. The values for this field are group specific, and the format is specified by the Nonce Type field. If no group-specific information is provided, the minimum length for this field is 4 bytes.

The payload type for the Nonce Payload is twelve (12).

7.12.2. Nonce Payload Processing

When processing the Nonce Payload, the following fields MUST be checked for correct values:

1. Next Payload, RESERVED, Payload Length - These fields are processed as defined in Section 7.2.2, "Generic Payload Header Processing".

2. Nonce Type - The Nonce Type value MUST be checked to be a valid nonce type as defined by Table 27. If the value is not valid, then an error is logged. If in Verbose Mode, an appropriate message containing notification value Payload-Malformed will be sent.

3. Nonce Data - This is the nonce data and it must be checked according to its content. The size of this field is defined in Section 7.12, "Nonce Payload". Refer to Section 5.2, "Group Establishment", for interpretation of this field.

8. GSAKMP State Diagram

Figure 28 presents the states encountered in the use of this protocol. Table 28 defines the states. Table 29 defines the transitions.

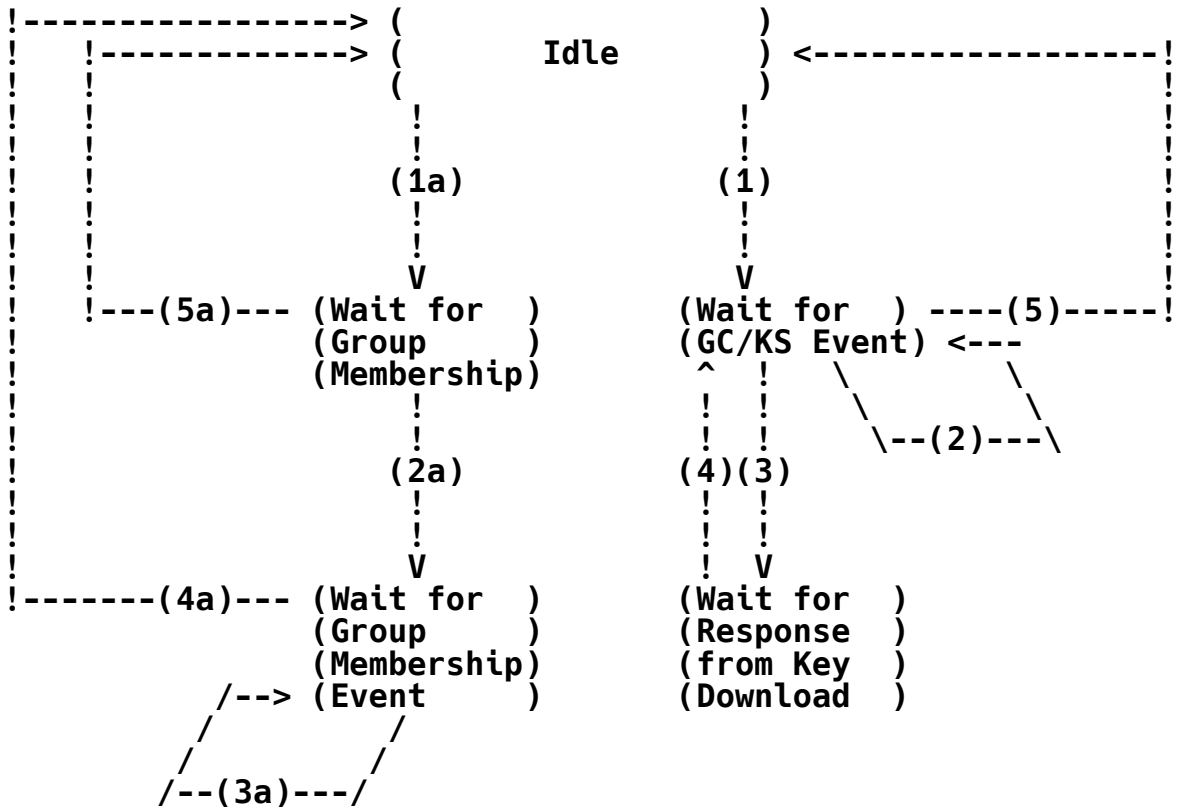

Figure 28: GSAKMP State Diagram

Table 28: GSAKMP States

Idle	GSAKMP Application waiting for input
Wait for GC/KS Event	GC/KS up and running, waiting for events
Wait for Response from Key Download	GC/KS has sent Key Download, waiting for response from GM
Wait for Group Membership	GM in process of joining group
Wait for Group Membership Event	GM has group key, waiting for group management messages.

Table 29: State Transition Events

Transition 1	Create group command
Transition 2	Receive bad RTJ Receive valid command to change group membership Send Compromise message x times Member Deregistration
Transition 3	Receive valid RTJ
Transition 4	Timeout Receive Ack Receive Nack
Transition 5	Delete group command
Transition 1a	Join group command
Transition 2a	Send Ack
Transition 3a	Receipt of group management messages
Transition 4a	Delete group command Deregistration command
Transition 5a	Time out Msg failure errors

9. IANA Considerations

9.1. IANA Port Number Assignment

IANA has provided GSAKMP port number 3761 in both the UDP and TCP spaces. All implementations MUST use this port assignment in the appropriate manner.

9.2. Initial IANA Registry Contents

The following registry entries have been created:

GSAKMP Group Identification Types (Section 7.1.1)
GSAKMP Payload Types (Section 7.1.1)
GSAKMP Exchange Types (Section 7.1.1)
GSAKMP Policy Token Types (Section 7.3.1)
GSAKMP Key Download Data Item Types (Section 7.4.1)
GSAKMP Cryptographic Key Types (Section 7.4.1.1)
GSAKMP Rekey Event Types (Section 7.5.1)
GSAKMP Identification Classification (Section 7.6.1)
GSAKMP Identification Types (Section 7.6.1)
GSAKMP Certificate Types (Section 7.7.1)
GSAKMP Signature Types (Section 7.8.1)
GSAKMP Notification Types (Section 7.9.1)
GSAKMP Acknowledgement Types (Section 7.9.1.1)
GSAKMP Mechanism Types (Section 7.9.1.3)
GSAKMP Nonce Hash Types (Section 7.9.1.3)
GSAKMP Key Creation Types (Section 7.11.1)
GSAKMP Nonce Types (Section 7.12.1)

Changes and additions to the following registries are by IETF Standards Action:

GSAKMP Group Identification Types
GSAKMP Payload Types
GSAKMP Exchange Types
GSAKMP Policy Token Types
GSAKMP Key Download Data Item Types
GSAKMP Rekey Event Types
GSAKMP Identification Classification
GSAKMP Notification Types
GSAKMP Acknowledgement Types
GSAKMP Mechanism Types
GSAKMP Nonce Types

Changes and additions to the following registries are by Expert Review:

GSAKMP Cryptographic Key Types
GSAKMP Identification Types
GSAKMP Certificate Types
GSAKMP Signature Types
GSAKMP Nonce Hash Types
GSAKMP Key Creation Types

10. Acknowledgements

This document is the collaborative effort of many individuals. If there were no limit to the number of authors that could appear on an RFC, the following, in alphabetical order, would have been listed: Haitham S. Cruickshank of University of Surrey, Sunil Iyengar of University Of Surrey Gavin Kenny of LogicaCMG, Patrick McDaniel of AT&T Labs Research, and Angela Schuett of NSA.

The following individuals deserve recognition and thanks for their contributions, which have greatly improved this protocol: Eric Harder is an author to the Tunneled-GSAKMP, whose concepts are found in GSAKMP as well. Rod Fleischer, also a Tunneled-GSAKMP author, and Peter Lough were both instrumental in coding a prototype of the GSAKMP software and helped define many areas of the protocol that were vague at best. Andrew McFarland and Gregory Bergren provided critical analysis of early versions of the specification. Ran Canetti analyzed the security of the protocol and provided denial of service suggestions leading to optional "cookie protection".

11. References

11.1. Normative References

[DH77] Diffie, W., and M. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, June 1977.

[FIPS186-2] NIST, "Digital Signature Standard", FIPS PUB 186-2, National Institute of Standards and Technology, U.S. Department of Commerce, January 2000.

[FIPS196] "Entity Authentication Using Public Key Cryptography," Federal Information Processing Standards Publication 196, NIST, February 1997.

[IKEv2] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC2412] Orman, H., "The OAKLEY Key Determination Protocol", RFC 2412, November 1998.

[RFC2627] Wallner, D., Harder, E., and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, June 1999.

[RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4514] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, June 2006.

[RFC4534] Colegrove, A. and H. Harney, "Group Security Policy Token v1", RFC 4534, June 2006.

11.2. Informative References

[BMS] Balenson, D., McGrew, D., and A. Sherman, "Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization", Work in Progress, February 1999.

[HCM] H. Harney, A. Colegrove, P. McDaniel, "Principles of Policy in Secure Groups", Proceedings of Network and Distributed Systems Security 2001 Internet Society, San Diego, CA, February 2001.

[HHMCD01] Hardjono, T., Harney, H., McDaniel, P., Colegrove, A., and P. Dinsmore, "Group Security Policy Token: Definition and Payloads", Work in Progress, August 2003.

[RFC2093] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification", RFC 2093, July 1997.

[RFC2094] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, July 1997.

[RFC2408] Maughan D., Schertler M., Schneider M., and Turner J., "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, Proposed Standard, November 1998

[RFC2451] Pereira, R. and R. Adams, "The ESP CBC-Mode Cipher Algorithms", RFC 2451, November 1998.

[RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, March 1999.

[RFC4523] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates", RFC 4523, June 2006.

[RFC2974] Handley, M., Perkins, C., and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, August 2001.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, March 2004.

[RFC4086] Eastlake, D., 3rd, Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

Appendix A. LKH Information

This appendix will give an overview of LKH, define the values for fields within GSAKMP messages that are specific to LKH, and give an example of a Rekey Event Message using the LKH scheme.

A.1. LKH Overview

LKH provides a topology for handling key distribution for a group rekey. It rekeys a group based upon a tree structure and subgroup keys. In the LKH tree shown in Figure 29, members are represented by the leaf nodes on the tree, while intermediate tree nodes represent abstract key groups. A member will possess multiple keys: the group traffic protection key (GTPK), subgroup keys for every node on its path to the root of the tree, and a personal key. For example, the member labeled as #3 will have the GTPK, Key A, Key D, and Key 3.

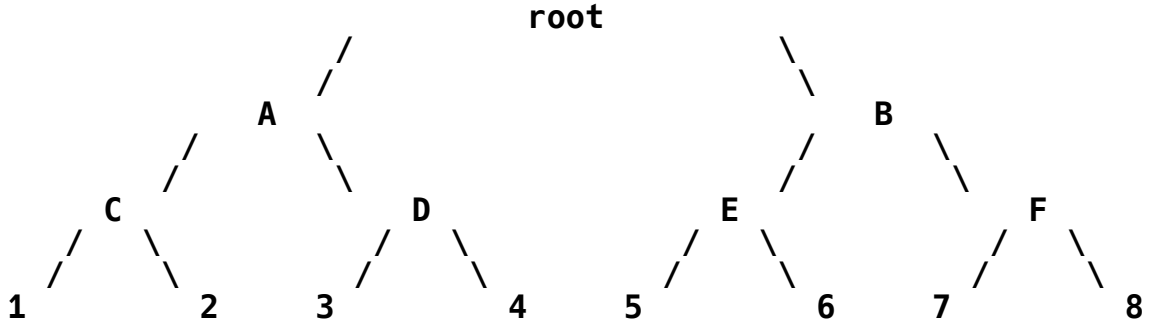

Figure 29: LKH Tree

This keying topology provides for a rapid rekey to all but a compromised member of the group. If Member 3 were compromised, the new GTPK (GTPK') would need to be distributed to the group under a key not possessed by Member 3. Additionally, new Keys A and D (Key A' and Key D') would also need to be securely distributed to the other members of those subtrees. Encrypting the GTPK' with Key B would securely distribute that key to Members 5, 6, 7, and 8. Key C can be used to encrypt both the GTPK' and Key A' for Members 1 and 2. Member 3's nearest neighbor, Member 4, can obtain GTPK', Key D', and Key A' encrypted under its personal key, Key 4. At the end of this process, the group is securely rekeyed with Member 3 fully excluded.

A.2. LKH and GSAKMP

When using LKH with GSAKMP, the following issues require attention:

1. Rekey Version # - The Rekey Version # in the Rekey Array of the Key Download Payload MUST contain the value one (1).

2. Algorithm Version - The Algorithm Version in the Rekey Event Payload MUST contain the value one (1).

3. Degree of Tree - The LKH tree used can be of any degree; it need not be binary.

4. Node Identification - Each node in the tree is treated as a KEK. A KEK is just a special key. As the rule stated for all keys in GSAKMP, the set of the KeyID and the KeyHandle MUST be unique. A suggestion on how to do this will be given in this section.

5. Wrapping KeyID and Handle - This is the KeyID and Handle of the LKH node used to wrap/encrypt the data in a Rekey Event Data.

For the following discussion, refer to Figure 30.

Key:
o: a node in the LKH tree
N: this line contains the KeyID node number
L: this line contains the MemberID number for all leaves ONLY

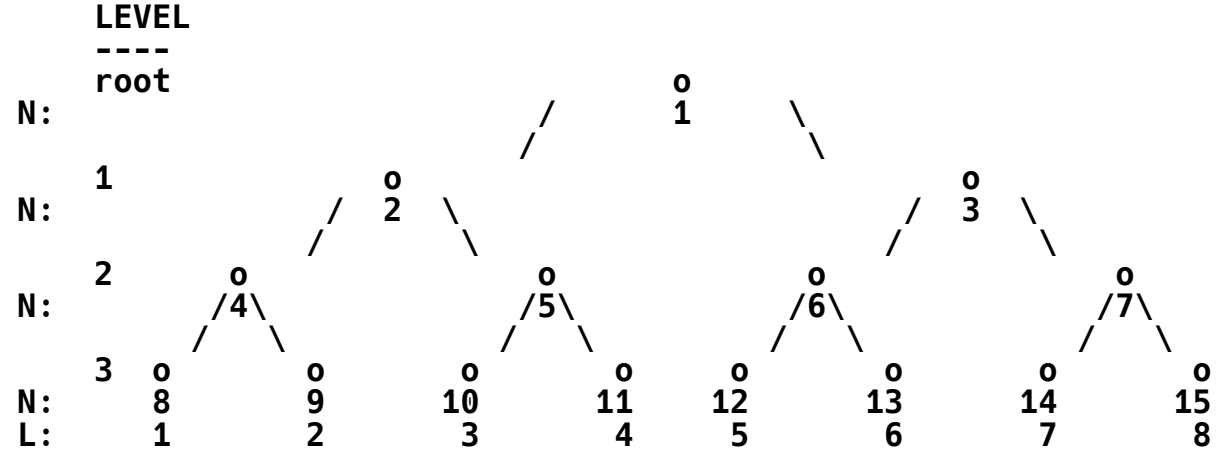

Figure 30: GSAKMP LKH Tree

To guarantee uniqueness of KeyID, the Rekey Controller SHOULD build a virtual tree and label the KeyID of each node, doing a breadth-first search of a fully populated tree regardless of whether or not the tree is actually full. For simplicity of this example, the root of the tree was given KeyID value of one (1). These KeyID values will be static throughout the life of this tree. Additionally, the rekey arrays distributed to GMs requires a MemberID value associated with them to be distributed with the KeyDownload Payload. These MemberID values MUST be unique. Therefore, the set associated with each leaf node (the nodes from that leaf back to the root) are given a MemberID. In this example, the leftmost leaf node is given MemberID value of one (1). These 2 sets of values, the KeyIDs (represented on lines N) and the MemberIDs (represented on line L), will give sufficient information in the KeyDownload and RekeyEvent Payloads to disseminate information. The KeyHandle associated with these keys is regenerated each time the key is replaced in the tree due to compromise.

A.3. LKH Examples

```
Definition of values:
0xLLLL          - length value
0xHHHHHHHH#     - handle value
YYYYMMDDHHMMSSZ - time value
```

A.3.1. LKH Key Download Example

This section will give an example of the data for the Key Download payload. The GM will be given MemberID 1 and its associated keys. The data shown will be subsequent to the Generic Payload Header.

```
| GTPK | MemberID 1 | KeyID 2 | KeyID 4 | KeyID 8

Number of Items                    - 0x0002
  Item #1:
    Key Download Data Item Type    - 0x00 (GTPK)
    Key Download Data Item Length  - 0xLLLL
      Key Type                     - 0x03 (3DES`CBC64`192)
      Key ID                       - KEY1
      Key Handle                   - 0xHHHHHHH0
      Key Creation Date            - YYYYMMDDHHMMSSZ
      Key Expiration Date          - YYYYMMDDHHMMSSZ
      Key Data                     - variable, based on key definition
  Item #2:
    Key Download Data Item Type    - 0x01 (Rekey - LKH)
    Key Download Data Item Length  - 0xLLLL
    Rekey Version Number           - 0x01
    Member ID                      - 0x00000001
```

```
      Number of KEK Keys            - 0x0003
        KEK #1:
          Key Type                  - 0x03 (3DES`CBC64`192)
          Key ID                    - 0x00000002
          Key Handle                - 0xHHHHHHH2
          Key Creation Date         - YYYYMMDDHHMMSSZ
          Key Expiration Date       - YYYYMMDDHHMMSSZ
          Key Data                  - variable, based on key definition
        KEK #2:
          Key Type                  - 0x03 (3DES`CBC64`192)
          Key ID                    - 0x00000004
          Key Handle                - 0xHHHHHHH4
          Key Creation Date         - YYYYMMDDHHMMSSZ
          Key Expiration Date       - YYYYMMDDHHMMSSZ
          Key Data                  - variable, based on key definition
        KEK #3:
          Key Type                  - 0x03 (3DES`CBC64`192)
          Key ID                    - 0x00000008
          Key Handle                - 0xHHHHHHH8
          Key Creation Date         - YYYYMMDDHHMMSSZ
          Key Expiration Date       - YYYYMMDDHHMMSSZ
          Key Data                  - variable, based on key definition
```

A.3.2. LKH Rekey Event Example

This section will give an example of the data for the Rekey Event payload. The GM with MemberID 6 will be keyed out of the group. The data shown will be subsequent to the Generic Payload Header.

```
| Rekey Event Type | GroupID | Date/Time | Rekey Type |
Algorithm Ver | # of Packets |
{ (GTPK)2, (GTPK, 3', 6')12, (GTPK, 3')7 }
```

This data shows that three packets are being transmitted. Read each packet as:

a) GTPK wrapped in LKH KeyID 2
b) GTPK, LKH KeyIDs 3' & 6', all wrapped in LKH KeyID 12
c) GTPK and LKH KeyID 3', all wrapped in LKH KeyID 7

NOTE: Although in this example multiple keys are encrypted under one key, alternative pairings are legal (e.g., (GTPK)2, (GTPK)3', (3')6', (3')7', (6')12).

We will show the format for all header data and packet (b).

```
   Rekey Event Type  - 0x01 (GSAKMP`LKH)
   GroupID           - 0xAABBCCDD
                       0x12345678
   Time/Date Stamp   - YYYYMMDDHHMMSSZ
   Rekey Event Type  - 0x01 (GSAKMP`LKH)
   Algorithm Vers    - 0x01
   # of RkyEvt Pkts  - 0x0003
   For Packet (b):
   Packet Length       - 0xLLLL
   Wrapping KeyID      - 0x000C
   Wrapping Key Handle - 0xHHHHHHHD
   # of Key Packages   - 0x0003
     Key Package 1:
       Key Pkg Type  - 0x00 (GTPK)
       Pack Length   - 0xLLLL
         Key Type            - 0x03 (3DES`CBC64`192)
         Key ID              - KEY1
         Key Handle          - 0xHHHHHHH0
         Key Creation Date   - YYYYMMDDHHMMSSZ
         Key Expiration Date - YYYYMMDDHHMMSSZ
         Key Data            - variable, based on key definition
     Key Package 2:
       Key Pkg Type  - 0x01 (Rekey  - LKH)
       Pack Length   - 0xLLLL
         Key Type            - 0x03 (3DES`CBC64`192)
         Key ID              - 0x00000003
         Key Handle          - 0xHHHHHHH3
         Key Creation Date   - YYYYMMDDHHMMSSZ
         Key Expiration Date - YYYYMMDDHHMMSSZ
         Key Data            - variable, based on key definition
     Key Package 3:
       Key Pkg Type  - 0x01 (Rekey  - LKH)
       Pack Length   - 0xLLLL
         Key Type            - 0x03 (3DES`CBC64`192)
         Key ID              - 0x00000006
         Key Handle          - 0xHHHHHHH6
         Key Creation Date   - YYYYMMDDHHMMSSZ
         Key Expiration Date - YYYYMMDDHHMMSSZ
         Key Data            - variable, based on key definition
```

Authors' Addresses

Hugh Harney (point-of-contact)
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, MD 21046

Phone: (443) 430-8032
Fax: (443) 430-8181
EMail: hh@sparta.com

Uri Meth
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, MD 21046

Phone: (443) 430-8058
Fax: (443) 430-8207
EMail: umeth@sparta.com

Andrea Colegrove
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, MD 21046

Phone: (443) 430-8014
Fax: (443) 430-8163
EMail: acc@sparta.com

George Gross
IdentAware Security
82 Old Mountain Road
Lebanon, NJ 08833

Phone: (908) 268-1629
EMail: gmgross@identaware.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).