SPARTA, Inc.
INTERNET-DRAFT
draft-harney-gkmp-spec-00.ps

Hugh Harney, Carl Muckenhirn, Thomas Rivers
SPARTA, Inc., Secure Systems Engineering Division
September, 1994

Group Key Management Protocol (GKMP) Specification

Abstract

Currently, protocols exist which create pairwise symmetric keys for encryption between any hosts on a network. With advances in multicast communication routing, group communication applications are being developed (video teleconferencing, white boards). These applications need grouped symmetric keys to secure their communications. Currently, grouped symmetric keys are not generated and automatically distributed among the communication group members.

This specification proposes a protocol to create grouped symmetric keys and distribute them amongst communication. This protocol has the following advantages: 1) virtually invisible to operator, 2) no central key distribution site is needed, 3) only group members have the key, 4) sender or receiver oriented operation, 5) can make use of multicast communications protocols.

Status of this memo

1 Background

Traditional key management distribution has mimicked the military paper based key accounting system. Key was distributed, ordered, and accounted physically leading to large lead times and expensive operations.

Cooperative key management algorithms exist that allow pairwise keys to be generated between two equipment's. This gives the a quicker more reliable key management structure capable of supporting large numbers of secure communications. Unfortunately, only pairwise keys are supported using these methods today.

This document describes a protocol for establishing and rekeying groups of cryptographic keys (more than two) on the internet. We refer to the approach as the Group Key Management Protocol (GKMP).

1.1 Protocol Overview

The GKMP creates key for cryptographic groups, distributes key to the group members, ensures (via peer to peer reviews) rule based access control of keys, denies access to known compromised hosts, and allow hierarchical control of group actions.

The key generation concept used by the GKMP is cooperative generation between two protocol entities. There are several key generation algorithms viable for use in the GKMP (i.e., RSA, Diffe-Hellman, elliptic curves). All these algorithms use asymmetric key technology to pass information between two entities to create a single cryptographic key.

The GKMP then distributes the group keys to qualified GKMP entities. This distribution process is a mutually suspicious process (all actions and identities must be verified).

The GKMP provides a peer to peer review process. Protocol entities pass permission certificates (PC) as part of the group key distribution process. The PCs contain access control information about a particular site. This access control information is assigned by a higher authority which then signs the PC. Therefor each entity can verify the permissions of any other GKMP entity but can modify none. Each protocol entity checks the permissions and compares them the level of service requested. If the permissions do not exceed or equal the request, the service is denied.

The GKMP supports compromise recovery. A list of compromised GKMP entities is distributed to group members during key management actions. In essence, a Compromise Recovery List (CRL) allows group members to drop connections with compromised entities. The GKMP delegates control of groups to specific group controllers so it will be somewhat easier to distribute the CRL to the most important GKMP entities. During each key management action the CRL version number is passed, when a CRL update is detected it is downloaded and verified (it is signed by a higher authority).

The GKMP allows control of group actions. In certain networks it is desirable for a higher authority to strictly control the generation of groups. These networks usually have a central network operations authority. The GKMP allows these authorities to remotely order group actions. These orders are signed by that authority and verified by all entities involved with the group.

The GKMP is an application layer protocol. It's independent of the underlying communication protocol. However, if multicast service is available it will speed the rekey of the cryptographic groups. Hence, the GKMP does use multicast services if they are available.

2 Overview: GKMP Roles

Creation and distribution of grouped key require assignment of roles. These identify what functions the individual hosts perform in the protocol. The two primary roles are those of key distributor and member. The controller initiates the creation of the key, forms the key distribution messages, and collects acknowledgment of key receipt from the receivers. The members wait for a distribution message, decrypt, validate, and acknowledge the receipt of new key.

2.1 Group controller

The group controller (GC) is the a group member with authority to perform critical protocol actions (i.e., create key, distribute key, create group rekey messages, and report on the progress of these actions). All group members have the capability to be a GC and could assume this duty upon assignment.

The GC helps the cryptographic group reach and maintain key synchronization. A group must operate on the same symmetric cryptographic key. If part of the group loses or inappropriately changes it's key, it will not be able to send or receive data to another host operating on the correct key. Therefor, it is important that those operations that create or change key are unambiguous and controlled (i.e., it would not be appropriate for multiple hosts to try to rekey a net simultaneously). Hence, someone has to be in charge – that is the controller.

2.2 Group member

Simply stated a group member is any group host who is not acting as the controller. The group members will: assist the controller in creating key, validate the controller authorization to perform actions, accept key from the controller, request key from the controller, maintain local CRL lists, perform peer review of key management actions, and manage local key.

3 Data Item primitives

3.1 Group members list:

In a sender oriented group, the GC must be given a list of net members. The controller will then initiate contact with these net members and create the group.

3.2 Group Token:

The group token is created by the authority which commands a group. The Token contains information the net members need to ensure a controller is authorized to create a group and exactly what constrains are intended to be places on the group. The group token contains the following fields: Group identification,

- GC ID,
- Group action (create, rekey, delete),
- Group permissions (rules to guide access control),
- Rekey interval (life span of group key),
- Token version (identifier to identify current token),
- Token signature (asymmetric signature using the group commanders private key),
- Group commanders public key (this public key is itself signed by the network security manager to bind the public to a specific net member ID).

3.3 Grp ID:

The group must be uniquely identified to allow for several different groups to coexist on a network.

3.4 GTEK ID:

Unique identifier of GTEK (can include state information).

3.5 GKEK ID:

Unique identifier of GKEK (can include state information).

3.6 GTEK creation field:

In a cooperative key creation protocol each party contributes some field used to create the key.

3.7 GKEK creation field:

In a cooperative key creation protocol each party contributes some field used to create the key.

3.8 Distributor signature:

Asymmetric signature using the GCs private key.

3.9 Distributor public key:

Public half of the GCs signature key pair. (this public key is itself signed by the network security manager to bind the public to a specific net member ID.

3.10 Member signature:

Asymmetric signature using the selected members private key.

3.11 Member public:

Public half of the selected members signature key pair. (this public key is itself signed by the network security manager to bind the public to a specific net member ID.

3.12 Controller permissions:

Controller permissions are assigned by the security manager. The security managers signature will bind the permissions to the controller ID.

3.13 SKEK ID:

This field identifies exactly which SKEK is being created. This allows multiple groups to interoperate on a net simultaneously.

3.14 SKEK creation field:

This field contains the information contributed for use in the KEK creation process.

3.15 Member permissions:

Member permissions are assigned by the security manager. The security managers signature will bind the permissions to the controller ID.

3.16 Encrypted Grp Keys:

This data item is encrypted in the KEK (session or group) created for the download of keys. It is the GTEK and GKEK created for a group. A checksum is also encrypted. This ensures the confidentiality and data integrity of the GTEK and GKEK.

3.17 Confirmation of decryption:

This is a short (byte) field indicating decryption of the message and exactly what type of message was decrypted.

3.18 Request:

A request field contains the specific request one net member may make to another. The requests range from (group join, CRL update, pairwise TEK generation, detection, group creation, status).

Member delete list:

A list of group members being administratively deleted from the group.

4 Message definitions

4.1 Command_Create Group:

This message contains the following data item primitives (Group members, Grp ID, Grp controller ID, Grp action, Grp permissions, Rekey interval, Token version, Token signature, Token public key). This message may be confidential due to the group permissions field. In sensitive systems it will need encryption prior to transmission.

4.2 Create Grp Keys_1:

This message passes the information needed to create the group keys from the GC to the selected net member. This message contains (Grp ID, Request, GTEK ID, GKEK ID, GTEK creation field, GKEK creation field, Grp token, Controller signature, Controller public)

4.3 Create Grp Keys_2:

This message passes the information needed to create the group keys from the selected net member to the GC. This message contains: (Grp ID, GTEK ID, GKEK ID, GTEK creation field, GKEK creation field, member signature, member public)

4.4 Negotiate Grp Keys_1:

This message passes the group token and GCs permissions to the selected net member. This information can be sensitive and needs to be protected. Therefor, this message is encrypted in the GTEK just created.

This encryption includes the appropriate data integrity checks. This message1 contains: (Grp ID, TEK ID, KEK ID, Group token, Controller permissions)

4.5 Negotiate Grp Keys_2:

This message passes the selected net members permissions to the GC. This message1 contains: (Grp ID, GTEK ID, GKEK ID, Member permissions). This information can be sensitive and needs to be protected. Therefor, this message is encrypted in the GTEK just created. This encryption includes the appropriate data integrity checks.

4.6 Create Session KEK_1:

This message sends information to create a KEK for one time use between the GC and selected net member.

4.7 Create Session KEK_2:

This message sends information to create a KEK for one time use between the selected net member and GC.

4.8 Negotiate Session Keys_1:

This message passes the group ID, SKEK ID, CRL version number, Group token and GCs permissions to the selected net member. This information can be sensitive and needs to be protected. Therefor, this message is encrypted. If an appropriate pairwise key is available then that key should be used. If not the KEK just created could be used to encrypt the message.

4.9 Negotiate Session Keys_2:

This message identifies the group, SKEK, CRL version number and the member permissions. This information can also be sensitive and needs protection.

4.10 Download Grp Keys:

This message includes a GRP ID and Encrypted Grp Keys data items.

4.11 Key download ack:

This message contains the GRP ID and Confirmation_decryption data items. It confirms the receipt and verified decryption of the GTEK and GKEK.

4.12 Rekey _Multicast:

This message contains: Grp ID, GTEK ID, GKEK ID, Group token, Controller permissions. The rekey message is encrypted in the GKEK already resident in all the group member sites. This leads to a single message capable of being accepted by all group members.

4.13 Request_Group_Join:

This message contains Request, Grp ID, Member Signature, Member Public.

4.14 Delete_Group_Keys:

This message contains: grp ID, Request, Member delete list, Controller signature, Controllers public.

4.15 Grp_Keys_Deleted_Ack:

This message contains (grp ID, member ID, member signature, member public.

4.16 Delete_Group_Keys:

This message contains (grp ID, request, member delete list, controller signature, controller public).

4.17 Grp_Keys_Deleted_Ack:

This message contains (grp ID, member ID, member signature, member public)

5 State definitions

There are thirteen separate states the in the protocol. They are described below:

5.1 State 1:

The source address is checked to ensure it is not on the CRL.

The token field is validated with the public key of the source.

The token version number is checked to ensure this token is current.

The group ID is checked to see if this group exists.

The controller ID field is then read. If the receiver is listed as the GC, the receiver assumes the role of controller. If not, the role assumed is that of receiver.

The GC reads the group permission field in the group token. It then verifies that its' personnel permissions exceed or equal those of the group.

The GC will creates its' portion of the key creation message.

The Create Grp Keys_1 message is completed and transmitted.

5.2 State 2:

The source signature field is validated using the public key of the source.

The source ID field is compared against the local CRL. If the source is on the CRL the association is terminated.

The request field is read. The local contributions to the group keys are created.

The Group keys are created and stored pending negotiation.

The key table is updated to show the group key pending negotiation.

5.3 State 3:

The permission certificate is retrieved and validated using the security managers public key. The permissions of the message source are checked to verify they meet or exceed those of the group.

The group token is retrieved and validated using the appropriate public key.

The token version number is checked to ensure the token is current.

The group ID specified in the token is compared with the actual group ID. If they are different the exchange is terminated.

The controller ID specified in the token is compared with the GC ID. If they do not match the exchange is terminated.

The local permissions are compared to the permissions specified for the group. If they do not meet or exceed the group permissions the exchange is terminated and a report is generated.

The rekey interval specified in the token is stored locally.

The key table is updated to reflect the key permissions, rekey interval, group ID and current time.

5.4 State 4:

The permission certificate is retrieved and validated using the security members public key. The permissions of the message source are checked to verify they meet or exceed those of the group.

The key table is updated to reflect the key permissions, rekey interval, group ID and current time.

5.5 State 5:

The source signature field is validated using the public key of the source.

The source ID field is compared against the local CRL. If the source is on the CRL, the association is terminated.

The request field is read. The local contribution to the SKEK are created. The SKEK is created and stored pending negotiation.

The key table is updated to show the SKEK pending negotiation.

5.6 State 6:

The permission certificate is retrieved and validated using the security managers public key. The permissions of the message source are checked to verify they meet or exceed those of the group.

The group token is retrieved and validated using the appropriate public key.

The token version number is checked to ensure the token is current.

The group ID specified in the token is stored.

The controller ID specified in the token is compared with the GC ID. If they do not match the exchange is terminated.

The local permissions are compared to the permissions specified for the group. If they do not meet or exceed the group permissions the exchange is terminated and a report is generated.

The rekey interval specified in the token is stored locally.

The key table is updated to reflect the key permissions, rekey interval, group ID and current time.

5.7 State 7:

The permission certificate is retrieved and validated using the security managers public key. The permissions of the message source are checked to verify they meet or exceed those of the group.

The key table is updated.

5.8 State 8:

The group ID is checked.

The group keys are decrypted using the SKEK. Data integrity checks are validated to ensure proper decryption.

The key table is updated to reflect the new group keys, key permissions, rekey interval, group ID and current time.

5.9 State 9:

Update group management log.

5.10 State 10:

The permission certificate is retrieved and validated using the security managers public key. The permissions of the message source are checked to verify they meet or exceed those of the group.

The group token is retrieved and validated using the appropriate public key.

The token version number is checked to ensure the token is current.

The group ID specified in the token is checked.

The controller ID specified in the token is compared with the GC ID. If they do not match the exchange is terminated.

The local permissions are compared to the permissions specified for the group. If they do not meet or exceed the group permissions the exchange is terminated and a report is generated.

The rekey interval specified in the token is stored locally.

The new group keys are decrypted with the current GKEK. The data integrity field is checked to ensure proper decryption.

The key table is updated to reflect the key permissions, rekey interval, group ID and current time.

5.11 State 11:

Validate signature using sources public key.

Check to see if member initiated group join is available. If not, ignore. If so begin distribution of group keys.

5.12 State 12:

Validate signature using GCs public.

Retrieve delete list. Check to see if on delete list, if so continue.

Create Grp_Keys_Deleted_Ack

Delete group keys

5.13 State 13:

Validate signature using GCs public.

Retrieve delete list. If list is global delete, verify alternative key.

Switch group operations to alternative key.

Create Grp_Keys_Deleted_Ack.

Delete group keys.

6 Functional Definitions–Group Key Management Protocol

The GKMP consists of multiple functions necessary to create, distribute, rekey and manage groups of symmetric keys. These functions are:

- Group creation (sender initiated group)
  - Create Group keys
  - Distribute Group keys
- Group rekey
  - Create Group keys
  - Rekey Group
- Member initiated join
- Group member delete

The following sections will describe each function, including data primitives and message constructs. The associated diagrams will represent the specifics (sequence, location and communications sources and destinations) of the messages and processes necessary.

6.1 Group creation

Member initialization is a three-step function that involves commanding the creation of the group, creation of the group keys and then distribution of those keys to "other" group members. Messages between the GC and the first member generate two keys for future group actions: the group traffic encryption key (GTEK) and the group key encryption key (GKEK). Messages between the GC and the other members are for the purpose of distributing the keys. These functions are described in the following sections.

6.1.1 Group command

The very first action is for some entity to command the group. This command is sent to the GC.

6.1.2 Create group keys

The first member must cooperate with the GC to create future group keys. Reliance on two separate hosts to create group keys maximizes the probability that the resulting key will have the appropriate cryptographic properties. A single host could create the key if the randomization function were robust and trusted. Unfortunately this usually requires specialized hardware not available at most host sites. The intent of this protocol was to utilize generic hardware to enhance the extendibility of the GKMP. Hence, cooperative key generation mechanisms are used.

To facilitate a well ordered group creation, management information must be passed between the controller and the group members. This information uniquely identifies the GC identity, it's permissions, authorization to create keys, the future groups permissions, current state of the compromise list, and management information pertaining to the keys being created. All this information is protected from forgery by asymmetric signature technologies. The public key used to verify net wide parameters (e.g., individual host permissions) are widely held. The public key to verify locally generated information, like peer identity, is sent with the messages. This alleviates the hosts public key storage requirements.

The goals of the key creation process are:

- cooperatively generate a GTEK and GKEK,
- allow the key creators to verify the identity of the key creation partner by verifying the messages signatures.
- share public keys
- allow validation of the GC, by signing the group identification, GC identification, and group permissions.
- send the group identity, GC identity, group member identities, group permissions, and group rekey interval to the first member, signed by the group commander (when the group was remotely commanded).

This function consists of four messages between the GC and the first member. The initial messages are for the establishment of the GTEK and GKEK. This is accomplished by the GC sending a signed

Net Controller	**Messages**	**Net Member B**
The Create Group command is received by net member A. State 1	<— Command-Create Group	
	Create Grp Keys_1—->	
		State 2
	<—-Create Grp Keys_2	
State 2		
	Negotiate Grp Keys_1——>	
		State 3
	<—-Negotiate Grp Keys_2	
State 4		

Figure 1: State Diagram: Create Group Keys

`Create_Group_Keys_1` message to the first member. This message contains two random values necessary to generate the GTEK and GKEK. This message also contains the public key of the GC.

The first member validates the signed `Create_Group_Keys_1` message, builds and sends a signed `Create_Group_Keys_2` message to the GC. He generates the GTEK and GKEK, and stores the received public key. The `Create_Group_Keys_2` message contains the random values necessary for the GC to generate the GTEK and GKEK. This message also contains the public key of the first member.

The GC validates the signed `Create_Group_Keys_2` message, generates the GTEK and GKEK, builds the `Negotiate_Group_Keys_1` message for transmission to the first member, and stores the received public key.

The GC sends the `Negotiate_Group_Keys_1` message to the first member encrypted in the GTEK that was just generated.

The first member decrypts the `Negotiate_Group_Keys_1` message and extracts the group identification, GC identification, group members, group permissions, key rekey interval, CRL version number, and certifying authority signature. The group identification, GC identification, and group permissions fields are validated based on the extracted group commanders signature (if this is a remotely commanded group this signature identifies the remote host). If these fields validate, the first members internal structures are updated.

6.1.3 Distributing Group Keys to Other Members

The other group members must get the group keys before the group is fully operational. The purpose of other group member initialization is as follows:

- cooperatively generate a session key encryption key (SKEK) for the transmission of the GTEK and GKEK from the GC,
- allow each member to verify the identify of the controller and visa versa,
- allow each member to verify the controllers authorization to create the group,
- send the key packet (KP) (consisting of the GTEK, GKEK), group identity, GC identity, group member identities, group permissions, and group rekey interval to the other members,

Net Controller	Messages	Net members, individually
	Create Session KEK_1—->	
		State 5
	<—-Create Session KEK_2	
State 5		
	Negotiate Session Keys_1—->	
		State 6
	<—-Negotiate Session Keys_2	
State 7		
	Download Grp Keys——>	
		State 8
	<—- Key download ack	
State 9		

Figure 2: State Diagram: Distribute Keys

This function consists of six messages between the GC and the other members. The initial messages are for the establishment of a SKEK. This is accomplished by the GC sending a signed `Create_Session_KEK_1` message to the other member. This message contains the random value necessary for the other member to generate the SKEK. This message also contains the public key of the GC.

The other member validates the `Create_Session_KEK_1` message, builds and sends a `Create_Session_KEK_2` message to the GC, generates the SKEK, and stores the received public key. The `Create_Session_KEK_2` message contains the random value necessary for the GC to generate the SKEK. This message also contains the public key of the other member.

The GC validates the `Create_Session_KEK_2` message, generates the SKEK, builds the `Negotiate_Session_KEK_1 message for transmission to the other member, and stores the received public key.`

The GC sends the `Negotiate_Session_KEK_1` message to the other member encrypted in the SKEK that was just generated. The `Negotiate_Session_KEK_1` message includes the group ID, group token, controller permissions, and CRL version number.

The other member decrypts the `Negotiate_Session_KEK_1` message, verifies the authority and identification of the controller, ensures the local CRL is up to date, and builds a `Negotiate_Session_KEK_2` message for transmission to the GC.

The GC receives the `Negotiate_Session_KEK_2` message and builds a `Download_Grp_Keys` message for transmission to the other member.

The GC sends the `Download_Grp_Keys` message to the other member encrypted in the SKEK that was just generated. (note: the key used to encrypt the negotiation messages can be combined differently to create the KEK.)

The other members decrypts the `Download_Grp_Keys` message and extracts the KP, group identification, GC identification, group members, group permissions, key rekey interval, and group commanders signature. The group identification, GC identification, and group permissions fields are validated based on the signature. If these fields validate, the other members internal key storage tables are updated with the new keys.

6.2 Group Rekey

Rekey is a two-step function that involves message exchange between the GC and a "first member" and "other members." Messages between the GC and the first member are exactly as described for group creation. Messages between the GC and the other members are for the purpose of distributing the new GTEK and the new GKEK. These functions are described in the following sections.

6.2.1 Create Group Keys

The first member function for a rekey operation is the same as that for key initialization. Please refer to the group creation section entitled "2.1 Create group keys".

6.2.2 Rekey

The purpose of rekey is as follows:

- send the new GTEK and new GKEK to the other members,
- allow each member to verify the identify of the controller,
- allow each member to verify the controllers authorization to rekey the group, group identification, and GC identification,
- send the group identity, GC identity, group member identities, group permissions, and group rekey interval to the other members,

The messages to create and negotiate the group keys are the same as stated during group creation. As such they have been omitted here.

The rekey portion of this function consists of one message between the GC and the other members. The GC builds a signed `Rekey_Multicast` message for transmission to the other member. As the name implies this message can be multicast to the entire group. The GC sends the signed `Rekey_Multicast` message to the other members encrypted in the current GKEK.

The other members decrypt and validate the signed `Rekey_Multicast` message and extract the new KP, group identification, GC identification, group members, group permissions, key rekey interval, and rekey command signature. The group identification, GC identification, and group permissions fields are validated based on the extracted rekey command signature. If these fields validate, the key database tables are updated.

6.3 Member Initiated Join

The GKMP will support member initiated joins to the group. This type of service is most attractive when the group initiator does not need to control group membership other than to verify that all members of the group conform to some previously agreed upon rules.

Net Controller	Messages	Net members, individually
The Create Group command is received by net member A.	<—– Command-Create Group	
State 1		
	Create Grp Keys_1—->	
		State 2
	<—–Create Grp Keys_2	
State 2		
	Negotiate Grp Keys_1——>	
		State 3
	<—–Negotiate Grp Keys_2	
State 4		
	Rekey _Multicast——->	
		State 10

Figure 3: State Diagram: Rekey

One example of this type of group is corporations job vacancies. A corporation may want to keep its job vacancies confidential and may decide to encrypt the announcements. The group creator doesn't care who gets the announcements as long as they are in the corporation. When an employee tries to access the information the GC looks at the employees permissions (signed by some higher authority). If they indicate the employee is part of the corporation the controller allows access to the group.

Before a potential group member can join group operations, they must request the key from the GC, unambiguously identify themselves, pass their permissions, and receive the keys. These require several messages to pass between GC and the joining member. The purpose of these messages are as follows:

- Request group join from controller
- cooperatively generate a SKEK for the transmission of the group traffic encryption and GKEK from the GC,
- allow each member to verify the identify of the controller and visa versa,
- allow each member to verify the controllers authorization to create the group,
- send the KP, group identity, GC identity, group member identities, group permissions, and group rekey interval to the other members,

The series of messages for a member initiated join is very similar to the series of messages to distribute group keys during group creation. In fact, the series are identical except for the addition of a request to join message sent from the joining member to the controller when the join is member initiated. This message should not require encryption since it probably does not contain sensitive information. However, in some military systems the fact that a member wants to join a group maybe sensitive from a traffic analysis viewpoint. In these specialized instances, a pairwise TEK may be created, if one does not already exist, to hide the service request.

This function consists of seven messages between the GC and the joining member. The first message is created by the joining member and sent to the GC. It simply request membership in the group from

Net Controller	Messages	Net Members, individually
	<—— Request_Group_Join	
State 11		
	Create Session KEK_1—->	
		State 5
	<—-Create Session KEK_2	
State 5		
	Negotiate Session Keys_1—->	
		State 6
	<—-Negotiate Session Keys_2	
State 7		
	Download Grp Keys——>	
		State 8
	<—- Key download ack	
State 9		

Figure 4: State Diagram: Member Join

the controller. The controller makes the decision whether to respond to the request based on the group parameters - membership limits, membership lists.

The next messages are for the establishment of a SKEK. This is accomplished by the GC sending a signed Create_Session_KEK_1 message to the other member. This message contains the random value necessary for the other member to generate the SKEK. This message also contains the public key of the GC.

The other member validates the `Create_Session_KEK_1` message, builds and sends a `Create_Session_KEK_2` message to the GC, generates the SKEK, and stores the received public key. The `Create_Session_KEK_2` message contains the random value necessary for the GC to generate the SKEK. This message also contains the public key of the other member.

The GC validates the `Create_Session_KEK_2` message, generates the SKEK, builds the `Negotiate_Session_KEK_1` message for transmission to the other member, and stores the received public key.

The GC sends the `Negotiate_Session_KEK_1` message to the other member encrypted in the SKEK that was just generated.

The other member decrypts the `Negotiate_Session_KEK_1` message and builds a `Negotiate_Session_KEK_2` message for transmission to the GC.

The GC receives the `Negotiate_Session_KEK_2` message and builds a Download_Grp_Keys message for transmission to the other member.

The GC sends the`Download_Grp_Keys` message to the other member encrypted in the SKEK that was just generated. (note: the key used to encrypt the negotiation messages can be combined differently to create the KEK.)

The other members decrypts the`Download_Grp_Keys` message and extracts the KP, group identification, GC identification, group members, group permissions, key rekey interval, and group commanders signature. The group identification, GC identification, and group permissions fields are validated based on the signature. If these fields validate, the other members internal key storage tables are updated with the new keys.

Net Controller	Messages	Net Members
	Delete_Group_Keys ——>	
		State 12
	<—— Grp_Keys_Deleted_Ack	
State 9		

Figure 5: State Diagram: Cooperative Delete

6.4 Member Deletion

There are two types of member deletion scenarios - cooperative and hostile. The cooperative deletion scenarios is the removal of a trusted group member for some management reason (i.e., reduce group size, prepare the member for a move). The hostile deletion usually results in a loss of secure state at the members site (i.e., compromise, equipment breakage).

The two scenarios present different challenges to the network. Minimization of network impact is paramount in the cooperative scenario. We would like to leave the key group intact and have confidence that removing the cooperative group member will have no impact on the security of future group operations. In the case of a hostile deletion, the goal is to return to a secure operating state as fast as possible. In fact there is a trade-off. We could eliminate the compromised group as soon as the compromise is discovered, but this may cripple an important asset. So security concerns need to be balanced with operational concerns.

6.4.1 Cooperative Deletion

The cooperative deletion function occurs between a trusted member and the GC. It results in a reliable deletion of the group key encryption and GTEKs at the deleted member. This deletion is intended to be an administrative function.

This function consists of two messages between the GC and the member. The GC sends the `Delete_Group_Keys` message to the group, encrypted in the GTEK. The message identifies the member(s) that need to delete the group keys. The member(s) decrypt the `Delete_Group_Keys` message, extract the group identification, check the deleted member list, deletes the group traffic and key encryption keys for that group, and build the `Group_Keys_Deleted_Ack` message for transmission to the GC.

The `Grp_Keys_Deleted_Ack` message is encrypted in the group traffic key. The GC receives the `Grp_Keys_Deleted_Ack` message, decrypts it, and updates the group definition.

6.4.2 Hostile Deletion (Compromise)

Hostile deletion occurs when a the group losses trust in a member. We assume that all keys resident at the members site have been lost. We also assume the member will not cooperate. Therefor, we must essentially create another group, minus the untrusted member, and transfer group operations to that new group. When the group losses trust in the controller, another controller must be appointed and then the hostile deletion process can proceed.

There are some security and operational management issues surrounding compromise recovery. The essence of the issues involve a tradeoff between operational continuity and security vulnerability. If a member is

Net Controller	Messages	Net Members
	Delete_Group_Keys ——>	
		State 13

Figure 6: State Diagram: Hostile Delete

found to be bad, from a security point of view all traffic on the network should stop. However, if that traffic is supporting a critical operation, the group may prefer to live with the security leak rather than interrupt the group communication.

The GKMP provides two mechanisms to help restrict access of compromised members. First, it implements a Certificate Revocation List (CRL) which is checked during the group creation process. Thus it will not allow a compromised member to be included in a new group. Second, the GKMP facilitates the creation of another group (minus the compromised member(s)). However, it does not dictate whether or not the group may continue to operate with a compromised member.

The mechanism the GKMP uses to remove a compromised member is to key that member out. This entails creating a new group, without the compromised member, and switching group operations. The old group is canceled by several multicasts of a group delete message.

This function consists of one message from the GC to all members. The GC sends the `Delete_Group` message to all members encrypted in the GTEK. This results in the deletion of the group traffic and key encryption keys in all group members. All members decrypt the received `Delete_Group` message, validate the authorization, extracts the group identification, and delete the group traffic and key encryption keys.

7 Addresses of Authors

Hugh Harney
SPARTA, Inc.
Secure Systems Engineering Division
9861 Broken Land Parkway, Suite 300
Columbia, MD 21046-1170
United States
telephone: +1 410 381 9400 (ext. 203)
electronic mail: `hh@columbia.sparta.com`

Carl Muckenhirn
SPARTA, Inc.
Secure Systems Engineering Division
9861 Broken Land Parkway, Suite 300
Columbia, MD 21046-1170
United States
telephone: +1 410 381 9400 (ext. 208)
electronic mail: `cfm@columbia.sparta.com`

Thomas Rivers

SPARTA, Inc.
Secure Systems Engineering Division
9861 Broken Land Parkway, Suite 300
Columbia, MD 21046-1170
United States
telephone: +1 410 381 9400 (ext. 205)
electronic mail: `thr@columbia.sparta.com`