<table>
<thead>
<tr>
<th>Networking Module</th>
<th>Service Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>smmsConvertToHostOrder</td>
<td>smmsGroupListener</td>
</tr>
<tr>
<td>smmsConvertToNetworkOrder</td>
<td>smmsNotifyLeave</td>
</tr>
<tr>
<td>smmsCreateGroupListener</td>
<td>smmsProcessGroupTimeOut</td>
</tr>
<tr>
<td>smmsCreateNetworkListener</td>
<td>smmsNetworkListener</td>
</tr>
<tr>
<td>smmsForwardJoinRequest</td>
<td>smmsServiceCRLAck</td>
</tr>
<tr>
<td>smmsGenerateCRLAck</td>
<td>smmsServiceCRLFrame</td>
</tr>
<tr>
<td>smmsGenerateCRLFrame</td>
<td>smmsServiceDispersalAck</td>
</tr>
<tr>
<td>smmsGenerateData</td>
<td>smmsServiceGroupData</td>
</tr>
<tr>
<td>smmsGenerateDispersalAck</td>
<td>smmsServiceGroupDispersal</td>
</tr>
<tr>
<td>smmsGenerateGroupDispersal</td>
<td>smmsServiceHandoverReply</td>
</tr>
<tr>
<td>smmsGenerateHandoverReply</td>
<td>smmsServiceHandoverRequest</td>
</tr>
<tr>
<td>smmsGenerateHandoverRequest</td>
<td>smmsServiceAliveAck</td>
</tr>
<tr>
<td>smmsGenerateAliveAck</td>
<td>smmsServiceJoinReply</td>
</tr>
<tr>
<td>smmsGenerateJoinReply</td>
<td>smmsServiceJoinRequest</td>
</tr>
<tr>
<td>smmsGenerateJoinRequest</td>
<td>smmsServiceLeaveReply</td>
</tr>
<tr>
<td>smmsGenerateLeaveRequest</td>
<td>smmsServiceLeaveRequest</td>
</tr>
<tr>
<td>smmsGenerateRejoinReply</td>
<td>smmsServiceRejoinReply</td>
</tr>
<tr>
<td>smmsGenerateRejoinRequest</td>
<td>smmsServiceRejoinRequest</td>
</tr>
<tr>
<td>smmsGenerateThirdHandshake</td>
<td>smmsServiceThirdHandshake</td>
</tr>
</tbody>
</table>
Figure 29 Group Management and Networking Modules

### Networking Module

#### smmsConvertToHostOrder

**Module:** Networking  

**Parameters:** struct reqMsg *rMsg
Returns: int status

Calls: None

Preconditions: rMsg is a valid pointer to a reqMsg structure.

Functionality: This function processes the numeric fields of a request message and converts it to host byte order.

smmsConvertToNetworkOrder

Module: Networking

Parameters: struct reqMsg *rMsg

Returns: int status

Calls: None

Preconditions: rMsg is a valid pointer to a reqMsg structure.

Functionality: This function processes the numeric fields of a request message and converts it to network byte order.

smmsCreateGroupListener

Module: Networking

Parameters: char * groupName, int port

Returns: int status

Calls: smmsGroupListener

Preconditions: The port value and name pair is unique among existing groups of this node.

Functionality: This function creates a new thread running the smmsGroupListener body, which will initialize the necessary data structures and perform the required protocol logic.

smmsCreateNetworkListener

Module: Networking

Parameters: None
**Returns:** int status

**Calls:** smmsNetworkListener

**Preconditions:** None

**Functionality:** The function smmsCreateNetworkListener creates a new thread to run the smmsNetworkListener function body.

---

**smmsForwardJoinRequest**

**Module:** Networking

**Parameters:** struct reqMsg rMsg, int sock, struct sockaddr_in ipAddress

**Returns:** int status

**Calls:** None

**Preconditions:** rMsg contains a JOIN_REQ to an unknown group.

**Functionality:** This function forwards a join request message to immediate neighbors and enters the Id and Sequence Number information into its records to prevent duplication.

---

**smmsGenerateCRLAck**

**Module:** Networking

**Parameters:** struct reqMsg *crlAck, char* groupName

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsGetGroupCounter, smmsSymEncryptData

**smmsSetGroupCounter**

**Preconditions:** rMsg contains a valid CRL Frame.
**Functionality:** smmsGenerateCRLAck generates an acknowledgement packet to indicate that a Certificate Revocation List has been received from the parent.

**smmsGenerateCRLFrame**

**Module:** Networking

**Parameters:** char* groupName, int nodeId, struct reqMsg* crlMesg

**Returns:** int status

**Calls:** smmsGetChildId, smmsGetChildKey, smmsGetGroupCounter, smmsSymEncryptData, smmsSetGroupCounter, smmsGenerateDigest, smmsAsymEncryptData

**Preconditions:** crlMsg has been initialized.

**Functionality:** A call to this function generates a Certificate Revocation List frame using the retrieved information from a parent. If this function is called at the source, this frame is generated over the stored revocation information. The detailed structure of this message can be found in section 2.4.7.

**smmsGenerateData**

**Module:** Networking

**Parameters:** struct struct reqMsg *rMsg, char* groupName, unsigned char *mesg, int mesgSize

**Returns:** int status

**Calls:** smmsGetChildId, smmsGetChildKey, smmsGetGroupCounter, smmsSymEncryptData, smmsSetGroupCounter

**Preconditions:** None

**Functionality:** smmsGenerateData is called by the application layer to generate a data packet to be sent to other group members.

**smmsGenerateDispersalAck**

**Module:** Networking
Parameters: struct reqMsg *dispAck, char* groupName

Returns: int status

Calls: smmsGetGroupKey, smmsGetGroupCounter, smmsSymEncryptData

smmsSetGroupCounter

Preconditions: rMsg contains a valid Group Dispersal message.

Functionality: Upon receiving a Group Dispersal message, a child node informs the parent by generating a Dispersal Acknowledgement message by calling the smmsGenerateDispersalAck function.

smmsGenerateGroupDispersal

Module: Networking

Parameters: struct reqMsg *dispMsg, char *groupName, int nodeId

Returns: int status

Calls: smmsGetChildId, smmsGetChildKey, smmsGetGroupCounter, smmsSymEncryptData, smmsSetGroupCounter

Preconditions: An application level call to disband the group has been received.

Functionality: A group source node can call this function to generate a Group Dispersal message, which will be relayed to all child nodes to inform them of the group cancellation.

smmsGenerateHandoverReply

Module: Networking

Parameters: struct reqMsg *handoffRep, char* groupName

Returns: int status

Calls: smmsGetGroupKey, smmsGetGroupCounter, smmsSymEncryptData

smmsSetGroupCounter

Preconditions: rMsg contains a valid Handover Request message.
**Functionality:** After a Handover Request message has been obtained and authentication with another group member is completed, this function is called to generate a Handover Reply message to inform the parent of the handover outcome.

**smmsGenerateHandoverRequest**

**Module:** Networking

**Parameters:** struct reqMsg *rMsg, char *groupName, int nodeId

**Returns:** int status

**Calls:** smmsGetChildId, smmsGetChildKey, smmsGetGroupCounter, smmsSymEncryptData, smmsSetGroupCounter

**Preconditions:** Application level request to leave has been received.

**Functionality:** Before a group leave occurs, child nodes must be requested to switch to another parent. This function is used to generate a packet declaring this intent.

**smmsGenerateIsAliveAck**

**Module:** Networking

**Parameters:** struct reqMsg *isAliveAck, char* groupName

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsGetGroupCounter, smmsSymEncryptData

**Preconditions:** rMsg contains a valid data packet.

**Functionality:** When a child node receives data, it replies by sending a Data Acknowledgement packet. This function generates the required acknowledgement structure.

**smmsGenerateJoinReply**
Module: Networking

Parameters: struct reqMsg *joinRep, char* groupName, struct sockaddr_in ipAddress, int groupPort

Returns: int status

Preconditions: Group Listener and Network Listener have been initialized.

Functionality: This function is used to construct a JOIN_REP message for a particular group.

smmsGenerateJoinRequest

Module: Networking

Parameters: struct reqMsg *rMsg, char* groupName

Returns: int status

Preconditions: Group Listener and Network Listener have been initialized.

Functionality: This function is used to construct a JOIN_REQ message for a particular group. This message is broadcast with controlled range to reach member nodes.

smmsGenerateLeaveReply

Module: Networking

Parameters: struct reqMsg *joinRep, char* groupName, struct sockaddr_in ipAddress, int groupPort

Returns: int status

Calls: smmsGetGroupKey, smmsGetGroupCounter, smmsSymEncryptData, smmsSetGroupCounter

Preconditions: Node is a member of the group, groupName.

Functionality: When a child node is given the leave command by the application layer, this function is called to generate the LEAVE_REQ message to be sent to the respective parent.
smmsGenerateLeaveRequest

Module: Networking

Parameters: struct reqMsg *joinRep, char* groupName, struct sockaddr_in ipAddress, int groupPort

Returns: int status

Calls: smmsGetGroupKey, smmsGetGroupCounter, smmsSymEncryptData

Preconditions: Node is a member of the group, groupName.

Functionality: When a child node is given the leave command by the application layer, this function is called to inform children of this intent and later on to send a LEAVE_REQ message.

smmsGenerateRejoinReply

Module: Networking

Parameters: struct reqMsg *rejoinRep, char *groupName, int nodeId

Returns: int status

Calls: smmsGetChildId, smmsGetChildKey, smmsGetGroupCounter, smmsSymEncryptData, smmsSetGroupCounter

Preconditions: rMsg contains a valid Rejoin Request message from a child node.

Functionality: When a child node attempts to reestablish lost communication, it issues a Rejoin Request message. Upon reception of this message, a node will call this function to generate a reply message.

smmsGenerateRejoinRequest

Module: Networking
**smmsGenerateRejoinRequest**

**Parameters:** struct reqMsg *rejoinReq, char* groupName

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsGetGroupCounter, smmsSymEncryptData

smmsSetGroupCounter

**Preconditions:** Node is a member of group, groupName.

**Functionality:** When communication is lost, a node will try to establish communication using a Rejoin Request message. The smmsGenerateRejoinRequest function generates the required message.

**smmsGenerateThirdHandshake**

**Module:** Networking

**Parameters:** struct reqMsg *rejoinRep, char *groupName, int nodeId

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsGetGroupCounter, smmsSymEncryptData

smmsSetGroupCounter

**Preconditions:** rMsg contains a valid Dispersal Acknowledgement message.

**Functionality:** The Third Handshake message is used to close group communication at a child node after a Disperse Group message has been received. This function creates the structure described in section 2.4.11.

**smmsGroupListener**

**Module:** Networking

**Parameters:** void *args

**Returns:** None

**Calls:** smmsServiceGroupData, smmsServiceLeaveRequest, smmsServiceLeaveReply, smmsServiceRejoinRequest, smmsServiceRejoinReply, smmsServiceCRLFrame, smmsServiceCRLAck, smmsServiceGroupDispersal, smmsServiceDispersalAck,
**smmsServiceThirdHandshake, smmsServiceHandoverRequest, smmsServiceHandoverReply, smmsServiceIsAliveAck**

**Preconditions:** The provided argument list contains a unique port value and name pair for this node. The network interface for the entire application is consistent.

**Functionality:** This function generates a socket and binds itself to the given port on the default network interface. It starts processing requests and replies targeted at this socket, which is the shared group port for members of this group.

**smmsNetworkListener**

**Module:** Networking

**Parameters:** void *args

**Returns:** None

**Calls:** smmsServiceJoinRequest, smmsServiceJoinReply

**Preconditions:** The UDP MASTER_PORT is not occupied.

**Functionality:** This function generates a socket and binds itself to the MASTER_PORT on the default network interface. It starts processing requests targeted at this socket, which is the shared network port for all nodes supporting the protocol.

**smmsNotifyLeave**

**Module:** Networking

**Parameters:** char* groupName

**Returns:** int status

**Calls:** smmsGenerateLeaveRequest, smmsConvertToNetworkOrder

**Preconditions:** All children have left the group.

**Functionality:** This function generates a leave request and forwards it to the parent. This function is called after all children have been requested to perform handover.

**smmsProcessTimeout**

**Module:** Networking
**Parameters:** struct eventElement* timeOutEvent, struct protectedEventQueue* timerQueue

**Returns:** int status

**Calls:** None

**Preconditions:** The event provided in timeoutEvent has timed out.

**Functionality:** This function processes timeout events. An event is described by a post condition and an identifier. The related message or event is found and processed accordingly.

---

**smmsServiceCRLAck**

**Module:** Networking

**Parameters:** struct reqMsg *rMsg

**Returns:** int status

**Calls:** None

**Preconditions:** rMsg contains a valid CRL Acknowledgement message. The node is in state SENDING_CRL.

**Functionality:** This function processes the CRL Acknowledgement message and verifies its authenticity. It removes existing timeout entries from the queue.

---

**smmsServiceCRLFrame**

**Module:** Networking

**Parameters:** struct reqMsg *rMsg

**Returns:** int status

**Calls:** smmsGetChildId, smmsGetChildKey, smmsGetGroupCounter, smmsSymDecryptData, smmsSetGroupCounter, smmsGenerateCRLAck

**Preconditions:** rMsg is a valid pointer to a CRL Frame. Node is in states RECEIVING_DATA or SENT_CRL_REQUEST.
**Functionality:** First this function verifies the counter and the validity of the CRL message. If an existing child node is within the CRL, it is removed. Further join requests by this node will be ignored.

**smmsServiceDispersalAck**

**Module:** Networking

**Parameters:** struct reqMsg *rMsg

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsGetGroupCounter, smmsSymDecryptData

smmsSetGroupCounter, smmsGenerateThirdHandshake

**Preconditions:** rMsg contains a valid Dispersal Acknowledgement message. The node is in state SENT_DISPERSAL.

**Functionality:** This function processes the validity of the Dispersal Message. If it is authentic, it will generate a Third Handshake message and remove the child node from the group.

**smmsServiceGroupData**

**Module:** Networking

**Parameters:** groupName, struct reqMsg * rMsg

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsSymDecryptData, smmsGenerateIsAliveAck

smmsReadDecryptedData, smmsServiceSignature

**Preconditions:** rMsg contains valid data. The node is in states AUTHENTICATED or RECEIVING_DATA.

**Functionality:** The function smmsServiceGroupData decrypts the data packet and determines its sub-type. If a signature message is included in the packet, it is processed further by the smmsServiceSignature function. Otherwise it is enqueued to be processed by the smmsReadDecryptedData function.

**smmsServiceGroupDispersal**

**Module:** Networking
Parameters: groupName, struct reqMsg * rMsg

Returns: int status

Calls: smmsGetGroupKey, smmsGetGroupCounter, smmsSymDecryptData

smmsSetGroupCounter, smmsGenerateDispersalAck

Preconditions: rMsg contains a valid Group Dispersal message.

Functionality: This function processes the Group Dispersal message and verifies its authenticity. An authentic dispersal message will be replied with an acknowledgement and upon reception of a Third Handshake message the group state will be deleted.

smmsServiceHandoverReply

Module: Networking

Parameters: groupName, struct reqMsg * rMsg

Returns: int status

Calls: smmsGetChildId, smmsGetChildKey, smmsGetGroupCounter,
smmsSymDecryptData, smmsSetGroupCounter, smmsRemoveChildNode

Preconditions: rMsg contains a valid Handover Reply message. The node is in state REQ_HANDOVER.

Functionality: Servicing a Handover Reply message occurs by verifying authenticity and removing the child node from the group. If this message exchange occurred due to a LEAVE_REQ, the group leave will be handled separately.

smmsServiceHandoverRequest

Module: Networking

Parameters: groupName, struct reqMsg * rMsg

Returns: int status

Calls: smmsGetGroupKey, smmsGetGroupCounter, smmsSymDecryptData

smmsSetGroupCounter, smmsGenerateHandoverReply, smmsCreateGroupListener

smmsJoinGroup
**Preconditions:** rMsg contains a valid Handover Request message. Node is in state RECEIVING_DATA.

**Functionality:** The Handover Request message is serviced by spawning a new thread to establish communication with another group member and monitoring the outcome of this join attempt. Upon successful completion of the join attempt, the previous group state is removed and an acknowledgement is generated by calling smmsGenerateHandoverReply.

---

**smmsServiceIsAliveAck**

**Module:** Networking

**Parameters:** groupName, struct reqMsg * rMsg

**Returns:** int status

**Calls:** smmsGetChildId, smmsGetChildKey, smmsGetGroupCounter, smmsSymDecryptData, smmsSetGroupCounter

**Preconditions:** rMsg contains a valid Data Acknowledgement message. The node is in states SENDING_DATA or AUTHENTICATED.

**Functionality:** Data acknowledgements are serviced by verifying authenticity and removing existing data timeout entries.

---

**smmsServiceJoinReply**

**Module:** Networking

**Parameters:** struct reqMsg rMsg, struct sockaddr_in ipAddress, struct protectedEventQueue* timerQueue
**Returns:** int status

**Calls:** smmsAddGroupState, smmsSetGroupParent, smmsCreateGroupListener

smmsGenerateAuthenticationRequest

**Preconditions:** rMsg is a valid JOIN_REP message. The node state is JOIN_REQ_SENT.

**Functionality:** A JOIN_REP message is created and authentication with the replying node starts. The group state is updated and a Group Listener is instantiated.

**smmsServiceJoinRequest**

**Module:** Networking

**Parameters:** struct reqMsg rMsg, struct sockaddr_in ipAddress, struct protectedEventQueue* timerQueue

**Returns:** int status

**Calls:** smmsGetGroupPort, smmsForwardJoinRequest

**Preconditions:** Group List has been initialized. rMsg is a valid JOIN_REQ message.

**Functionality:** This function processes the JOIN_REQ message and if it is not a recognized group, it forwards this message through smmsForwardJoinRequest. Otherwise it will reply with a JOIN_REP message and generate a state for this node with the SENT_JOIN_REP value.

**smmsServiceLeaveReply**

**Module:** Networking

**Parameters:** struct reqMsg rMsg, char* groupName, int sock, struct sockaddr_in ipAddress, struct protectedEventQueue* timerQueue

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsGetGroupCounter, smmsSymDecryptData

smmsSetGroupCounter, smmsRemoveGroupState

**Preconditions:** rMsg contains a valid Leave Reply message. The node is in state REQUESTED_LEAVE.
**Functionality:** This function processes the LEAVE_REP message and verifies its authenticity. If it is an authentic message, the group state will be removed.

**smmsServiceLeaveRequest**

**Module:** Networking

**Parameters:** struct reqMsg rMsg, char* groupName, int sock, struct sockaddr_in ipAddress, struct protectedEventQueue* timerQueue

**Returns:** int status

**Calls:** smmsGetChildId, smmsGetChildKey, smmsGetChildCounter, smmsSymDecryptData, smmsSymEncryptData, smmsRemoveChildNode

**Preconditions:** rMsg contains a valid Leave Request message.

**Functionality:** This function processes the LEAVE_REQ message and verifies its authenticity. If it is an authentic message, a LEAVE_REP message will be issued and the child state will be removed.

**smmsServiceRejoinReply**

**Module:** Networking

**Parameters:** struct reqMsg rMsg, char* groupName, int sock, struct sockaddr_in addr

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsGetGroupCounter, smmsSymDecryptData

smmsSetGroupCounter

**Preconditions:** rMsg contains a valid Rejoin Reply message. The node is in state SENT_REJOIN.

**Functionality:** If the Rejoin Request message has been accepted by the verified parent node, the group state will be maintained and normal operation will continue.

**smmsServiceRejoinRequest**

**Module:** Networking

**Parameters:** struct reqMsg rMsg, char* groupName, int sock, struct sockaddr_in addr

**Returns:** int status
**Calls:** smmsGetChildId, smmsGetChildKey, smmsGetChildCounter, smmsSymDecryptData, smmsSymEncryptData

**Preconditions:** rMsg contains a valid Rejoin Request message. The node is in state SUSPENDED.

**Functionality:** In order for this message to be processed properly, the parent must have the child node in the SUSPENDED state. If the message can be verified, the parent will issue a Rejoin Reply message and continue normal operation by switching to the SENDING_DATA state.

---

**smmsServiceThirdHandshake**

**Module:** Networking

**Parameters:** struct reqMsg rMsg, char* groupName, int sock, struct sockaddr_in addr

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsGetGroupCounter, smmsSymDecryptData

smmsSetGroupCounter

**Preconditions:** rMsg contains a valid Handshake message. The node is in state GROUP_CLOSING.

**Functionality:** Upon receiving a Third Handshake message, a node will verify authenticity and remove its group state.

---

**Group Management Module**
smmsAddChildNode

Module: Group Management

Parameters: struct groupState *state, unsigned char* key, int port, struct sockaddr_in addr

Returns: int status

Calls: None

Preconditions: The group state list has been initialized. State is a valid group state

Functionality: The smmsAddChildNode function introduces a new child to the group. It creates a new node; sets IP information; sets the key and returns. In case the group or the child does not exist, SMMS_FAILURE is returned.

smmsAddGroupState

Module: Group Management

Parameters: int type, char *groupName, int port

Returns: int status

Calls: None

Preconditions: The group state list has been initialized.

Functionality: The smmsAddGroupState function generates a new group by determining the next available port and initializing the parent-child relationship according to the type parameter. This new group is added to the Group State List.

smmsCreateGroup

Module: Group Management

Parameters: char *groupName, int port

Returns: int status

Calls: smmsAddGroupState, smmsCreateGroupListener

Preconditions: None
**Functionality:** This function calls `smmsAddGroupState` to create a new group state and spawns a new thread through `smmsStartGroupListener` to establish a Group Listener.

---

**smmsDisplayChildren**

**Module:** Group Management

**Parameters:** `char *groupName`, `struct groupNode *childList`

**Returns:** `int status`

**Calls:** `smmsGetGroup`

**Preconditions:** Group List has been initialized.

**Functionality:** This is an auxiliary method for the application layer, through which it can determine child node information for any given node.

---

**smmsDisplayParent**

**Module:** Group Management

**Parameters:** `char *groupName`, `struct groupNode *parentNode`

**Returns:** `int status`

**Calls:** `smmsGetGroupParent`

**Preconditions:** None

**Functionality:** This method is used by the application layer to determine the parent of a given node.

---

**smmsForwardGroupData**

**Module:** Group Management

**Parameters:** `char *groupName`, `struct reqMsg *rMsg`

**Returns:** `int status`
Calls: smmsGetChildKey, smmsSymEncryptData

Preconditions: The group state list has been initialized

Functionality: This function forwards received valid group data to registered child nodes.

smmsGetChild

Module: Group Management

Parameters: char* groupName, int id

Returns: struct groupNode * childNode

Calls: None

Preconditions: The group state list has been initialized

Functionality: smmsGetChild returns a pointer to the child node identified by id. If the child or the group do not exist the error code SMMS_FAILURE is returned.

smmsGetChildCounter

Module: Group Management

Parameters: int id, int* childCounter

Returns: int status

Calls: None

Preconditions: The group state list has been initialized

Functionality: smmsGetChildCounter returns the counter value from the child’s security association structure. If the child or the group do not exist the error code SMMS_FAILURE is returned.

smmsGetChildEncKey

Module: Group Management

Parameters: char* groupName, int id, char *childKey
**Returns**: int status

**Calls**: None

**Preconditions**: The group state list has been initialized

**Functionality**: This function returns the encryption key for a child node. If the child or the group do not exist the error code, SMMS_FAILURE is returned.

**smmsGetChildId**

**Module**: Group Management

**Parameters**: char* groupName, struct sockaddr_in ipAddress

**Returns**: int id

**Calls**: None

**Preconditions**: The group state list has been initialized

**Functionality**: The function smmsGetChildId returns the id of the child given the group’s name and its IP address. The child’s id is used for all operations on the child. If the child or the group do not exist the error code, SMMS_FAILURE is returned.

**smmsGetChildMacKey**

**Module**: Group Management

**Parameters**: char* groupName, int id, char *childKey

**Returns**: int status

**Calls**: None

**Preconditions**: The group state list has been initialized

**Functionality**: This function returns the message authentication key for a child node. If the child or the group do not exist the error code, SMMS_FAILURE is returned.

**smmsGetData**

**Module**: Group Management

**Parameters**: char *groupName, char *data, int dataSize
**Returns:** int status

**Calls:** smmsMesgQueueProtectedGet

**Preconditions:** Group List has been initialized.

**Functionality:** This function tests if a new data message is waiting in the group queue. If so, the message will be returned through the data pointer, otherwise it returns SMMS_FAILURE to indicate this condition. The queue size will be decremented by one.

---

**smmsGetGroup**

**Module:** Group Management

**Parameters:** char *groupName

**Returns:** struct groupState *Group

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** smmsGetGroup locates the group state with the given name and returns a pointer to it. It does not alter the group state list.

**smmsGetGroupCounter**

**Module:** Group Management

**Parameters:** char* groupName

**Returns:** int counter

**Calls:** None
**Preconditions:** The group state list has been initialized

**Functionality:** smmsGetGroupCounter returns the counter value for a particular group’s security association structure. If the group does not exist the error code SMMS_FAILURE is returned.

**smmsGetGroupEncKey**

**Module:** Group Management

**Parameters:** char* groupName, unsigned char* groupKey

**Returns:** int status

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The function smmsGetGroupEncKey returns a pointer to the encryption key of the group with the given name. For a parent this is the key shared with a child, and is unique for each child. For children, this is the key they share with their group parents. If the group does not exist, the error code, SMMS_FAILURE is returned.

**smmsGetGroupMacKey**

**Module:** Group Management

**Parameters:** char* groupName, unsigned char* groupKey

**Returns:** int status

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The function smmsGetGroupMacKey returns a pointer to the message authentication key of the group with the given name. If the group does not exist, the error code, SMMS_FAILURE is returned.
**smmsGetGroupParent**

**Module:** Group Management

**Parameters:** char *groupName

**Returns:** struct groupNode * parent

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The smmsGetGroupParent function returns a pointer to the parent of the group with the given name. If the group does not exist the error code SMMS_FAILURE is returned.

**smmsGetGroupPort**

**Module:** Group Management

**Parameters:** char *groupName

**Returns:** int port

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The function smmsGetGroupPort locates the group state within the Group State List and returns the group port for that particular group. If the group does not exist, a SMMS_FAILURE is returned.

**smmsGetGroupState**

**Module:** Group Management

**Parameters:** char *groupName

**Returns:** int groupState

**Calls:** None

**Preconditions:** The group state list has been initialized
**Functionality:** The function smmsGetGroupState locates the group within the Group State List and returns the group state for that particular group. If the group does not exist, a SMMS_FAILURE is returned.

**smmsGetParentPort**

**Module:** Group Management

**Parameters:** char *groupName

**Returns:** int groupPort

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The function smmsGetParentPort locates the group within the Group State List and returns the communication port established with that groups parent. If the group does not exist, a SMMS_FAILURE is returned.

**smmsJoinGroup**

**Module:** Group Management

**Parameters:** char *groupName

**Returns:** int status

**Calls:** smmsGenerateJoinRequest

**Preconditions:** The node is not a member of the given group.

**Functionality:** This function broadcasts a JOIN_REQ message and enters the JOIN_REQ_STATE into the Request Storage Table.
**smmsLeaveGroup**

**Module:** Group Management

**Parameters:** char *groupName

**Returns:** int status

**Calls:** smmsGenerateHandoverRequest, smmsServiceHandoverReply, smmsGenerateLeaveRequest, smmsServiceLeaveReply, smmsRemoveGroupState

**Preconditions:** Group List has been initialized.

**Functionality:** The node sends a LEAVE_REQ message to its parent, declaring its intent to leave the group. After this step it will enter the REQUESTED_LEAVE state.

**smmsReadDecryptedData**

**Module:** Group Management

**Parameters:** char *groupName, int struct reqMsg *rMsg

**Returns:** int status

**Calls:** smmsSymDecryptData, smmsMesgQueueProtectedPut

**Preconditions:** The passed message contains valid data.

**Functionality:** A call to smmsReadDecryptedData first retrieves the encryption key for the given group and decrypts the data in place using this key. After the message has been decrypted, it is enqueued for display in the group queue.

**smmsRemoveAllChildNodes**

**Module:** Group Management

**Parameters:** struct groupState *state

**Returns:** int status

**Calls:** None

**Preconditions:** The group state list has been initialized. Pointed group is valid.
**Functionality:** A call to `smmsRemoveAllChildNodes` removes all members of a given group. This function is used to disband a group.

**smmsRemoveAllGroups**

**Module:** Group Management

**Parameters:** None

**Returns:** int status

**Calls:** `smmsRemoveAllChildNodes`, `smmsMesgQueueProtectedGet`

**Preconditions:** The group state list has been initialized.

**Functionality:** The function `smmsCleanGroupState` removes all members of the Group State List.

**smmsRemoveChildNode**

**Module:** Group Management

**Parameters:** char * groupName, int nodeId

**Returns:** int status

**Calls:** None

**Preconditions:** The group state list has been initialized.

**Functionality:** Calling this function removes a member of the given group. This function is used to reset any state information between a parent and a child.

**smmsRemoveGroupState**

**Module:** Group Management

**Parameters:** char *groupName

**Returns:** int status

**Calls:** `smmsMesgQueueProtectedGet`, `smmsRemoveAllChildNodes`

**Preconditions:** The group state list has been initialized.
**Functionality:** The function `smmsRemoveGroupState` removes the given group's state from the Group State List, otherwise it returns an error.

**smmsSendToGroup**

**Module:** Group Management

**Parameters:** `char *groupName`, `unsigned char *msg`, `int msgSize`

**Returns:** `int status`

**Calls:** `smmsGetGroup`, `smmsForwardGroupData`

**Preconditions:** Group List has been initialized.

**Functionality:** This function retrieves the group state information and sends a message to all members of this group through `smmsForwardGroupData`.

**smmsSetBranch**

**Module:** Group Management

**Parameters:** `char* groupName`, `int id`, `int childCounter`

**Returns:** `int status`

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The `smmsSetBranch` function sets the initial counter value in a node’s security association structure. In case the group or the child does not exist `SMMS_FAILURE` is returned.

**smmsSetBranchEncKey**

**Module:** Group Management

**Parameters:** `char* groupName`, `int id`, `unsigned char* key`

**Returns:** `int status`

**Calls:** None

**Preconditions:** The group state list has been initialized
**Functionality:** The `smmsSetChildEncKey` function sets the encryption key in a node’s security association structure. In case the group or the child does not exist `SMMS_FAILURE` is returned.

**smmsSetChildEncKey**

**Module:** Group Management

**Parameters:** `char* groupName, int id, unsigned char* key`

**Returns:** `int status`

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The `smmsSetChildEncKey` function sets the message authentication key in a node’s security association structure. In case the group or the child does not exist `SMMS_FAILURE` is returned.

**smmsSetChildState**

**Module:** Group Management

**Parameters:** `char* groupName, int id, int newState`

**Returns:** `int status`

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The `smmsSetChildState` function performs a group state change for a node. In case the group or the child does not exist `SMMS_FAILURE` is returned.

**smmsSetGroupCounter**

**Module:** Group Management

**Parameters:** `char *groupName, int counter`

**Returns:** `int status`

**Calls:** None

**Preconditions:** The group state list has been initialized
Functionality: This function sets the initial counter value in a node’s security association structure for a group. In case the group or the child does not exist SMMS_FAILURE is returned.

smmsSetGroupEncKey

Module: Group Management

Parameters: char *groupName, char* key

Returns: int status

Calls: None

Preconditions: The group state list has been initialized

Functionality: The smmsSetGroupEncKey call changes the encryption key value of the given group with the given parameters. If the given group does not exist SMMS_FAILURE is returned.

smmsSetGroupMacKey

Module: Group Management

Parameters: char *groupName, char* key

Returns: int status

Calls: None

Preconditions: The group state list has been initialized
**Functionality:** The `smmsSetGroupMacKey` call changes the message authenticating key value of the given group with the given parameters. If the given group does not exist SMMS_FAILURE is returned.

**smmsSetGroupParent**

**Module:** Group Management

**Parameters:** char *groupName, int port, unsigned char *key, unsigned char *certHash, struct sockaddr_in ipAddress

**Returns:** int status

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The `smmsSetGroupParent` function changes the parent value of the given group with the given parameters. If the given group does not exist SMMS_FAILURE is returned.

**smmsSetGroupState**

**Module:** Group Management

**Parameters:** char* groupName, int newState

**Returns:** int status

**Calls:** None

**Preconditions:** The group state list has been initialized

**Functionality:** The `smmsSetGroupParent` function changes the parent value of the given group with the given parameters. If the given group does not exist SMMS_FAILURE is returned.

**smmsSetParentPort**

**Module:** Group Management

**Parameters:** char *groupName, int newPort

**Returns:** int status

**Calls:** `smmsGetGroup`
**Preconditions:** Group List has been initialized.

**Functionality:** This function sets the communication port value with a group’s parent node. After this operation, the group will be reached through this port.
Authentication Module

smmsGenerateAuthenticationReply

Module: Authentication

Parameters: char* groupName, int nodeId, struct reqMsg* authRep, struct certificate* selfCert, struct certificate* nodeCert, unsigned char* recvRandVal

Returns: int status

Calls: smmsConvertIntToChars, smmsAsymEncryptData, smmsSignDigest

Preconditions: Node is in state SENT_JOIN_REP.

Functionality: This function generates an authentication reply message as described in section 2.4.3 and returns the message for forwarding.

smmsGenerateAuthenticationRequest

Module: Authentication

Parameters: char* groupName, struct reqMsg* authReq,struct certificate* selfCert,int commPort

Returns: int status

Calls: smmsGenerateRandomNumber

Preconditions: A group with the provided name exists. There is a valid parent within the group. A valid join reply has been received.

Functionality: The function smmsGenerateAuthenticationRequest generates an authentication request and forwards it to the proposed group parent. After the exchange of authentication messages, either authentication fails, resulting in resetting of the group state or successful authentication. In the case of success, a key will be established between the parent and the child, the group state will be updated and the node will advance its state to AUTHENTICATED.
**smmsGenerateCRLRequest**

**Module:** Authentication

**Parameters:** struct reqMsg *crlReq, char* groupName

**Returns:** int status

**Calls:** smmsGetGroupKey, smmsGetGroupCounter, smmsSymEncryptData

**Preconditions:** Node is in state RECEIVING_DATA and didn’t receive a CRL Frame longer than a timeout period.

**Functionality:** This function generates an explicit request for a CRL message if after a timeout period no CRL message has been received.

---

**smmsGenerateKeySetup**

**Module:** Authentication

**Parameters:** char* groupName, int nodeId, struct reqMsg* keySetup, struct certificate* selfCert, struct certificate* nodeCert, unsigned char* recvRandVal

**Returns:** int status

**Calls:** smmsConvertIntToChars, smmsAsymEncryptData, smmsSignDigest

**Preconditions:** destPublicKey is a valid RSA public key.
**smmsGenerateKeyVerification**

**Functionality:** This function accepts the necessary parameters and constructs a KEY_SETUP message.

**Module:** Authentication

**Parameters:** `char* groupName, int nodeId, struct reqMsg* keyVerf`

**Returns:** `int status`

**Calls:** `smmsGetChildId, smmsGetChildKey, smmsSymEncryptData, smmsSetChildCounter`

**Preconditions:** The structure addr contains the address of an authenticating child and the node is in state SENT_AUTH_REQ.

**Functionality:** This function generates a key verification message and enters the counter value into the node structure of the authenticating child. The node proceeds to state KEY_SETUP_SENT.

**smmsGenerateSignature**

**Module:** Authentication

**Parameters:** `struct reqMsg *sigMsg, char* groupName`

**Returns:** `int status`

**Calls:** `smmsGetChildId, smmsGetChildKey, smmsSymEncryptData, smmsSetChildCounter, smmsGenerateDigest, smmsAsymEncryptData, smmsGenerateRandomNumber`

**Preconditions:** Either a valid signature message has been received or the signature period has been reached.

**Functionality:** This function builds a signature message over the data, which has been sent since the last time a signature message was generated. The signature message is described in detail in section 2.4.6.

**smmsGenerateSignatureAck**

**Module:** Authentication

**Parameters:** `struct reqMsg *sigAck, char* groupName`
Returns: int status

Calls: smmsGetGroupKey, smmsSymEncryptData, smmsSetGroupCounter

Preconditions: rMsg contains a valid Signature message. The node is in state RECEIVING_DATA.

Functionality: Received Signature messages are processed by calling this function and generating acknowledgement messages which are sent to the parent.

**smmsServiceAuthenticationReply**

**Module:** Authentication

**Parameters:** struct authMsg *rep, struct authMsg *setup, int sock, struct sockaddr_in addr, unsigned char *chanum, unsigned char *session_key, struct certificate mycert

**Returns:** int status

**Calls:** smmsAsymDecryptData, smmsConvertIntToChars, smmsVerifyCertificate

smmsVerifySignedDigest

**Preconditions:** Node is in state SENT_AUTH_REQ

**Functionality:** This function verifies an Authentication Reply and prepares a KEY_SETUP message by calling smmsGenerateKeySetup.

**smmsServiceAuthenticationRequest**

**Module:** Authentication

**Parameters:** reqMsg rMsg, int sock, struct sockaddr_in addr

**Returns:** int status

**Calls:** smmsGenerateRandomNumber, smmsVerifyCertificate, smmsSymEncryptData, smmsGenerateDigest

**Preconditions:** The node is a member of the given group. A valid join request has been received.
**Functionality:** The `smmsServiceAuthenticationRequest` function generates an Authentication Reply by verifying the certificate, generating a random challenge, encrypting a key contribution and signing the digest of this message.

**smmsServiceCRLRequest**

**Module:** Authentication

**Parameters:** `struct reqMsg rMsg, int sock, struct sockaddr_in addr`

**Returns:** `int status`

**Calls:** `smmsGetChildId, smmsGetChildKey, smmsSymEncryptData, smmsSetChildCounter, smmsGenerateCRLFrame`

**Preconditions:** `rMsg` contains a valid CRL Request message.

**Functionality:** A CRL Request message causes the parent to generate another CRL message based on the existing data and forward this message to the child node.

**smmsServiceKeySetup**

**Module:** Authentication

**Parameters:** `struct reqMsg rMsg, char *groupName, int sock, struct sockaddr_in addr, unsigned char *challenge, unsigned char *randomNum, unsigned char *sessionKey, struct certificate selfCert`

**Returns:** `int status`

**Calls:** `smmsAddChildNode, smmsVerifySignedDigest, smmsAsymDecryptData, smmsSetChildCounter`

**Preconditions:** The node is in state SENT_AUTH_REP.

**Functionality:** This function decrypts the key contribution, verifies the digest and adds the child node into the group.

**smmsServiceKeyVerification**

**Module:** Authentication

**Parameters:** `struct reqMsg rMsg`
**Returns:** int status

**Calls:** smmsSymDecryptData, smmsSetGroupNonce, smmsGenerateDigest

**Preconditions:** rMsg contains a key verification message.

**Functionality:** This function checks a key verification message and enters the counter value into the group structure to establish a security association.

**smmsServiceSignature**

**Module:** Authentication

**Parameters:** reqMsg rMsg, struct rsaPublicKey sourcePublicKey

**Returns:** int status

**Calls:** smmsVerifySignedDigest, smmsGetGroupKey, smmsSymDecryptData, smmsSetGroupCounter

**Preconditions:** The node is in state RECEIVING_DATA.

**Functionality:** Signature messages are serviced by inspecting integrity, verifying coverage of all received packages and checking freshness. If the signature packet is new, the signature table is reset and an acknowledgement is sent after the signature id is stored. This function also handles incorrect signature events.

**smmsServiceSignatureAck**

**Module:** Authentication

**Parameters:** reqMsg rMsg

**Returns:** int status

**Calls:** smmsGetChildId, smmsGetChildKey, smmsSymDecryptData, smmsSetGroupCounter

**Preconditions:** The node is in state SENDING_SIGNATURE.

**Functionality:** If a signature acknowledgement can be verified to be authentic, the signature period is updated and the operation is resumed by moving the child node to the SENDING_DATA state.
Cryptography Tools Module

smmsAsymDecryptData

Module: Cryptography Tools

Parameters: char* key, char *mesg, int messageLength

Returns: int status

Calls: None

Preconditions: mesg has been encrypted with the RSA algorithm using the given key.

Functionality: This function uses the RSA algorithm to decrypt a given message using the designated key.

smmsAsymEncryptData

Module: Cryptography Tools

Parameters: char* key, char *mesg, int messageLength

Returns: int status

Calls: None

Preconditions: messageLength is smaller than the key size.

Functionality: This function uses the RSA algorithm to encrypt a given message. It returns SMMS_FAILURE, if the given message is not conforming.

smmsCompareKey

Module: Cryptography Tools

Parameters: struct rsaPublicKey *key1, struct rsaPublicKey *key2

Returns: int predicate
Preconditions: key1 and key2 are valid RSA keys

Functionality: This function compares RSA keys and returns the comparison result.

**smmsConvertCharsToInt**

Module: Cryptography Tools

Parameters: unsigned char *data

Returns: int value

Preconditions: Data is a valid integer representation in char array format.

Functionality: This function converts an integer represented as a char into the integer format.

**smmsConvertIntToChars**

Module: Cryptography Tools

Parameters: int number, unsigned char *data

Returns: int status

Preconditions: None

Functionality: This function converts an integer into char notation.

**smmsCopyKey**

Module: Cryptography Tools

Parameters: rsaPublicKey *key1, rsaPublicKey *key2

Returns: int status

Calls: None

Preconditions: key1 is a valid RSA key
**Functionality:** This function copies RSA keys.

---

**smmsDeriveKeys**

**Module:** Cryptography Tools

**Parameters:** char* masterKey, char *macKey, char *encKey,

**Returns:** int status

**Calls:** smmsGenerateDigest

**Preconditions:** The value masterKey has been initialized.

**Functionality:** A call to smmsDeriveKeys generates two keys from a given seed. The details of the derivation can be found in Figure 27. It returns pointers to these generated keys.

---

**smmsGenerateCertificate**

**Module:** Cryptography Tools

**Parameters:** struct certificate *cert, int id, int seqNum, int type, int issuedTime, int expirationTime, struct rsaPrivateKey certAuthPrivateKey

**Returns:** int status

**Calls:** smmsConvertIntToChars, smmsSignDigest

**Preconditions:** None

**Functionality:** This function generates a valid SMMS certificate for future use. This function will be called offline by the Certificate Authority.

---

**smmsGenerateDigest**

**Module:** Cryptography Tools
Parameters: char *mesg, int mesgSize, char *digest

Returns: int status

Calls: None

Preconditions: None

Functionality: This function computes a SHA1 digest over the given message.

**smmsGenerateKeyPair**

Module: Cryptography Tools

Parameters: int keySize, struct rsaPublicKey* publicKey, struct rsaPrivateKey* privateKey

Returns: int status

Calls: None

Preconditions: The value keySize conforms with the RSA standard.

Functionality: The smmsGenerateKeyPair function prepares a public, private key pair for future use.

**smmsGenerateRandomNumber**

Module: Cryptography Tools
Parameters: unsigned char *randNum, int numberSize

Returns: int status

Calls: None

Preconditions: The random number generation system has been initialized.

Functionality: This function generates a cryptographically secure random number.

smmsSignDigest

Module: Cryptography Tools

Parameters: char *mesg, int mesgSize, char *signature, struct rsaPrivateKey* privateKey

Returns: int status

Calls: smmsAsymEncryptData, smmsGenerateDigest

Preconditions: None

Functionality: This integrated function computes the SHA1 digest of the message and signs it with the given key.

smmsSymDecryptData

Module: Cryptography Tools

Parameters: char* key, char *mesg, int mesgSize, int dataSize

Returns: int status

Calls: None

Preconditions: The key is of correct size.

Functionality: This function decrypts the given symmetrically encrypted data block in place. It requires the key and the size of the data.

smmsSymEncryptData

Module: Cryptography Tools

Parameters: char* key, char *mesg, int mesgSize
Returns: int dataSize

Calls: None

Preconditions: The key is of correct

Functionality: This function is based on the AES and encrypts the given data block in place. It returns the data length of the encrypted block.

smmsVerifyCertificate

Module: Cryptographic Tools

Parameters: struct reqMsg rMsg, struct rsaPublicKey *publicKey

Returns: int status

Calls: None

Preconditions: rMsg contains a valid authentication message.

Functionality: This function verifies whether the certificate contained within the authentication message is valid or not. This function is used both for authentication requests and authentication replies.

smmsVerifySignedDigest

Module: Cryptography Tools

Parameters: char *inputData, int inputSize, char *orgMesg, int msgSize, struct rsaPublicKey* publicKey

Returns: int predicate

Calls: smmsGenerateDigest, smmsAsymDecryptData

Preconditions: None

Functionality: This function decrypts a given message and compares the provided digest with the message.

Queue Management Module
smmsGetTimeOfDay

Module: Queue Management

Parameters: None

Returns: double timeVal

Calls: None

Preconditions: System time measurements are consistent

Functionality: This function returns a timestamp value obtained from the system.

smmsEventQueueAcquire

Module: Queue Management

Parameters: struct protectedEventQueue *fifo

Returns: int status

Calls: None

Preconditions: fifo is a valid eventQueue structure

Functionality: This function locks the requested queue.

smmsEventQueueEmpty

Module: Queue Management

Parameters: struct eventQueue *fifo

Returns: int predicate

Calls: None

Preconditions: fifo is a valid eventQueue structure

Functionality: This function returns whether the given queue is empty.

smmsEventQueueFull

Module: Queue Management
Parameters: struct eventQueue *fifo

Returns: int predicate

Calls: None

Preconditions: fifo is a valid eventQueue structure

Functionality: This function returns whether the given queue is full. In its current implementation, the queue size is not bounded and this function returns false when memory is available.

smmsEventQueueGet

Module: Queue Management

Parameters: struct eventQueue *fifo, struct eventElement *returnElement

Returns: int status

Calls: None

Preconditions: fifo is a valid timerQueue structure and contains at least one element

Functionality: This function allocates space for a replication of the retrieved element and copies fields. After this operation it removes the queue element and reduces queue size.

smmsEventQueueGetByTime

Module: Queue Management

Parameters: struct eventQueue *fifo, struct eventElement *returnElement

Returns: int status

Calls: smmsGetTimeOfDay

Preconditions: fifo is a valid eventQueue structure and contains at least one element.

Functionality: This function first tries to locate one entry with a timed out value. The time out criterion is calculated using the current time and TIMEOUT_VAL parameter. After that it allocates space for a replication of the retrieved element and copies fields. After this operation it removes the queue element and reduces queue size.
**smmsEventQueueInit**

**Module:** Queue Management

**Parameters:** None

**Returns:** struct eventQueue * fifo

**Calls:** None

**Preconditions:** None

**Functionality:** This function initializes a queue by allocating memory and setting control variables.

**smmsEventQueueProtectedDestroy**

**Module:** Queue Management

**Parameters:** struct protectedEventQueue * fifo

**Returns:** int status

**Calls:** smmsEventQueueEmpty, smmsEventQueueGet

**Preconditions:** fifo is a valid protectedEventQueue structure

**Functionality:** This function returns the allocated memory for an event queue object.

**smmsEventQueueProtectedEmpty**

**Module:** Queue Management
**Parameters:** struct protectedEventQueue *fifo

**Returns:** int predicate

**Calls:** smmsEventQueueEmpty

**Preconditions:** fifo is a valid protectedEventQueue structure

**Functionality:** This function tests whether the given queue is empty. It performs the test operation under mutual exclusion.

**smmsEventQueueProtectedFull**

**Module:** Queue Management

**Parameters:** struct protectedEventQueue *fifo

**Returns:** int predicate

**Calls:** smmsEventQueueFull

**Preconditions:** fifo is a valid protectedEventQueue structure

**Functionality:** This function returns whether the given queue is full. In its current implementation, the queue size is not bounded and this function's return value is equivalent to the unavailability of memory. It performs the test operation under mutual exclusion.

**smmsEventQueueProtectedGet**

**Module:** Queue Management

**Parameters:** struct protectedEventQueue *fifo

**Returns:** struct eventElement * ReturnElement

**Calls:** smmsEventQueueGet

**Preconditions:** fifo is a valid protectedEventQueue structure

**Functionality:** This function first obtains a lock on the given protected queue. If the given queue is not empty, it will issue a get command and retrieve the first element as ReturnElement from the queue.

**smmsEventQueueProtectedGetByTime**
Module: Queue Management

Parameters: struct protectedEventQueue *fifo

Returns: struct eventElement* ReturnElement

Calls: smmsEventQueueGetByTime

Preconditions: fifo is a valid protectedEventQueue structure

Post conditions: If fifo is not empty it will contain one less element.

Functionality: This function first obtains a lock on the given protected queue. If the given queue is not empty, it will issue a get command and retrieve the first element, which has timed out as ReturnElement from the queue.

smmsEventQueueProtectedInit

Module: Queue Management

Parameters: None

Returns: struct protectedEventQueue * fifo

Calls: smmsEventQueueInit

Preconditions: None

Post conditions: Memory is allocated for a protectedEventQueue structure.

Functionality: This function initializes a queue and the respective monitor structure to establish mutual exclusion. It returns a pointer to the structure.

smmsEventQueueProtectedPut

Module: Queue Management
Parameters: struct protectedEventQueue *fifo, struct eventElement *nextElement

Returns: int status

Calls: smmsEventQueuePut

Preconditions: fifo is a valid protectedEventQueue structure, nextElement is a valid queue element

Functionality: This function first obtains a lock on the given protected queue. If the given queue is not full, it will issue a put command and insert the element given by newElement into the queue.

smmsEventQueuePut

Module: Queue Management

Parameters: struct eventQueue *fifo, struct eventElement *nextElement

Returns: int status

Calls: None

Preconditions: fifo is a valid protectedEventQueue structure, nextElement is a valid queue element

Functionality: This function first obtains a lock on the given protected queue. If the given queue is not full, it will issue a put command and insert the element given by newElement into the queue.

smmsEventQueueRelease

Module: Queue Management

Parameters: struct protectedEventQueue *fifo

Returns: int status

Calls: None

Preconditions: fifo is a valid locked queue.

Functionality: This function releases the obtained lock on a queue object.
smmsEventQueueRemoveById

Module: Queue Management

Parameters: struct eventQueue *fifo, int ownerId, int ownerType, int prevState, char* groupName, int msgType

Returns: int status

Calls: None

Preconditions: fifo is a valid eventQueue structure

Functionality: This function first tries to locate the given entry with the given parameters. If this element can be found, it is removed from the queue and the queue size is reduced.

smmsEventQueueSchedule

Module: Queue Management

Parameters: struct protectedEventQueue* timeOutQueue, struct reqMsg rMsg, char* groupName, int timeOutUnits, int ownerId, int ownerType, int prevState, int finalState, int failState

Returns: int status

Calls: smmsEventQueueProtectedPut, smmsGetTimeOfDay

Preconditions: fifo is a valid eventQueue structure

Functionality: This function inserts a timeout event into the timeout queue given by timeOutQueue.

smmsMesgQueueEmpty
Module: Queue Management

Parameters: struct mesgQueue *fifo

Returns: int predicate

Calls: None

Preconditions: fifo is a valid mesgQueue structure

Functionality: This function returns whether the given queue is empty.

smmsMesgQueueFull

Module: Queue Management

Parameters: struct mesgQueue *fifo

Returns: int predicate

Calls: None

Preconditions: fifo is a valid mesgQueue structure

Functionality: This function returns whether the given queue is full. In its current implementation, the queue size is not bounded and this function returns false when memory is available.

smmsMesgQueueGet

Module: Queue Management

Parameters: struct mesgQueue *fifo, struct mesgElement *returnElement

Returns: int status
Calls: None

Preconditions: fifo is a valid mesgQueue structure and contains at least one element.

Functionality: This function allocates space for a replication of the retrieved element and copies fields. After this operation it removes the queue element and reduces queue size.

smmsMesgQueueInit

Module: Queue Management

Parameters: None

Returns: struct mesgQueue * fifo

Calls: None

Preconditions: None

Functionality: This function initializes a queue by allocating memory and setting control variables.

smmsMesgQueueProtectedDestroy

Module: Queue Management

Parameters: struct protectedMesgQueue *fifo

Returns: int predicate

Calls: smmsMesgQueueEmpty, smmsMesgQueueGet

Preconditions: fifo is a valid protectedMesgQueue structure

Functionality: This function returns the allocated memory for a given message queue.

smmsMesgQueueProtectedEmpty

Module: Queue Management

Parameters: struct protectedMesgQueue *fifo

Returns: int predicate

Calls: smmsMesgQueueEmpty
**Preconditions:** fifo is a valid protectedMesgQueue structure

**Functionality:** This function tests whether the given queue is empty. It performs the test operation under mutual exclusion.

---

**smmsMesgQueueProtectedFull**

**Module:** Queue Management

**Parameters:** struct protectedMesgQueue *fifo

**Returns:** int predicate

**Calls:** smmsMesgQueueFull

**Preconditions:** fifo is a valid protectedMesgQueue structure

**Functionality:** This function returns whether the given queue is full. In its current implementation, the queue size is not bounded and this function's return value is equivalent to the unavailability of memory. It performs the test operation under mutual exclusion.

**smmsMesgQueueProtectedGet**

**Module:** Queue Management

**Parameters:** struct protectedMesgQueue *fifo
Returns: struct mesgElement * ReturnElement

Calls: smmsMesgQueueGet

Preconditions: fifo is a valid protected queue structure

Functionality: This function first obtains a lock on the given protected queue. If the given queue is not empty, it will issue a get command and retrieve the first element as ReturnElement from the queue.

smmsMesgQueueProtectedInit

Module: Queue Management

Parameters: None

Returns: struct protectedMesgQueue * fifo

Calls: smmsMesgQueueInit

Preconditions: None

Functionality: This function initializes a queue and the respective monitor structure to establish mutual exclusion. It returns a pointer to the structure.

smmsMesgQueueProtectedPut

Module: Queue Management

Parameters: struct protectedMesgQueue *fifo, struct mesgElement *nextElement

Returns: int status

Calls: smmsMesgQueuePut

Preconditions: fifo is a valid protectedMesgQueue structure, nextElement is a valid queue element

Functionality: This function first obtains a lock on the given protected queue. If the given queue is not full, it will issue a put command and insert the element given by newElement into the queue. The queue size will increase by one.
**smmsMesgQueuePut**

**Module:** Queue Management

**Parameters:** struct msgQueue *fifo, struct msgElement *nextElement

**Returns:** int status

**Calls:** None

**Preconditions:** fifo is a valid protectedMsgQueue structure, nextElement is a valid queue element

**Functionality:** This function first obtains a lock on the given protected queue. If the given queue is not full, it will issue a put command and insert the element given by newElement into the queue, increasing the queue size by one.

**smmsTimerThreadLogic**

**Module:** Queue Management

**Parameters:** void * args

**Returns:** int status

**Calls:** smmsEventQueueAcquire, smmsEventQueueGetByTime, smmsProcessTimeOut, smmsEventQueueRelease

**Preconditions:** The parameter args contains a valid pointer to a protectedEventQueue.

**Functionality:** This is the function body, which performs the timeout management. It first obtains a lock on the time out queue and then processes the time out events by calling the function smmsProcessTimeOut.