SNMP Management: SNMPv3

OBJECTIVES

- *SNMPv3 features*
  - *Documentation architecture*
  - *Formalized SNMP architecture*
  - *Security*
- *SNMP engine ID and name for network entity*
- *SNMPv3 applications and primitives*
- *SNMP architecture*
  - *Integrates the three SNMP versions*
  - *Message-processing module*
  - *Dispatcher module*
  - *Future enhancement capability*
- *User security model, USM*
  - *Derived from user ID and password*
  - *Authentication*
  - *Privacy*
  - *Message timeliness*
- *View-based access control model, VACM*
  - *Configure set of MIB views for agent with contexts*
  - *Family of subtrees in MIB views*
  - *VACM process*

SNMPv2 was released, after much controversy, as a community-based SNMP framework, SNMPv2C, without any enhancement to security. SNMPv3 was subsequently developed to fulfill that need in SNMP management. Fortunately, SNMPv3 has ended up addressing more than just security. It is a framework for all three versions of SNMP. It is designed to accommodate future development in SNMP management with minimum impact to existing management entities. Modular architecture and documentation have been proposed to accomplish this goal.

The latest set of additional documentation [RFC 3410–3418, 3584] detailing the specifications of SNMPv3 is described in the RFCs listed in Table 7.1. They comprise IETF-adopted standards STD 62.

RFC 3410 provides an overview of SNMPv3 Framework.

Table 7.1 SNMPv3 RFCs

RFC 3410	Introduction and Applicability Statements (not STD)
RFC 3411	Architecture for Describing SNMP Management Frameworks
RFC 3412	Message Processing and Dispatching for SNMP
RFC 3413	SNMPv3 Applications
RFC 3414	User-based Security Model (USM) for SNMPv3
RFC 3415	View-based Access Control Model for SNMP
RFC 3416	Version 2 of the Protocol Operations for SNMP
RFC 3417	Transport Mappings for SNMP
RFC 3418	MIB for SNMP
RFC 3584	SNMPv3 Coexistence and Transition (BCP (Best Current Practices) 74)

RFC 3411 presents an overview of SNMPv3. It defines a vocabulary for describing SNMP Management Frameworks and an architecture for describing the major portions of SNMP Management Frameworks.

RFC 3412 describes message processing and dispatching for SNMP messages. Procedures are specified for processing multiple versions of SNMP messages to the proper SNMP Message Processing Models (MPM) and for dispatching packet data units (PDUs) to SNMP applications. A new MPM for version 3 is proposed in this document.

RFC 3413 defines five types of SNMP applications: command generators, command responders, notification originators, notification receivers, and proxy forwarders. It also defines the Management Information Base (MIB) modules for specifying targets of management operations, for notification filtering, and for proxy forwarding.

RFC 3414 addresses the User-based Security Model (USM) for SNMPv3 and specifies the procedure for providing SNMP message level security. MIBs for remotely monitoring and managing configuration parameters are also specified.

RFC 3415 is concerned with the Access Control Model that deals with the procedure for controlling access to management information. MIB is specified for remotely managing the configuration parameters for the view-based access control model (VACM).

RFC 3416 defines version 2 syntax and procedures for sending, receiving, and processing of SNMP PDUs.

RFC 3417 defines the transport of SNMP messages over various protocols.

RFC 3418 describing the behavior of an SNMP entity obsoletes RFC 1907 MIB for SNMPv2.

RFC 3584 describes the coexistence of SNMPv3, SNMPv2, and SNMPv1. It also describes how to convert MIB modules from SMIv1 format to SMIv2 format.

7.1 SNMPv3 KEY FEATURES

One of the key features of SNMPv3 is the modularization of architecture and documentation. The design of the architecture integrated SNMPv1 and SNMPv2 specifications with the newly proposed SNMPv3. This enables continued usage of legacy SNMP entities along with SNMPv3 agents and manager. That is good news as there are tens of thousands of SNMPv1 and SNMPv2 agents in the field.

An SNMP engine is defined with explicit subsystems that include dispatch and message-processing functions. It manages all three versions of SNMP to coexist in a management entity. Application services and primitives have been explicitly defined in SNMPv3. This formalizes the various messages that have been in use in the earlier versions.

Another key feature is the improved security feature. The configuration can be set remotely with secured communication that protects against modification of information and masquerade by using encryption schemes. It also tries to ensure against malicious modification of messages by reordering and time delaying of message streams, as well as protects against eavesdropping of messages.

The access policy used in SNMPv1 and SNMPv2 is continued and formalized in the access control in SNMPv3, designated VACM. The SNMP engine defined in the architecture checks whether a specific type of access (read, write, create, notify) to a particular object (instance) is allowed.

7.2 SNMPv3 DOCUMENTATION ARCHITECTURE

The numerous SNMP documents have been organized by IETF to follow a document architecture. The SNMP document architecture addresses how existing documents and new documents could be designed to be autonomous and, at the same time, be integrated to describe the different SNMP frameworks. The representation shown in Figure 7.1 reflects the contents of the specifications, but it is another perspective of what is given in RFC [3410]. It can be correlated with what we presented in Figure 4.4. Two sets of documents are of general nature. One of them is the set of documents on roadmap, applicability statement, and coexistence and transition.

The other set of documents, SNMP frameworks, comprises the three versions of SNMP. An SNMP framework represents the integration of a set of subsystems and models. A model describes a specific design of a subsystem. The implementation of an SNMP entity is based on a specific model based on a specific framework. For example, a message in an SNMP manager is processed using a specific message processing mode (we will discuss these later) in a specific SNMP3 framework. The SNMP frameworks document set is not explicitly shown in the pictorial presentation in RFC [2271], as we have done here. RFC [1901] in SNMPv2 and RFC [2271] in SNMPv3 are SNMP framework documents.

The information model and MIBs cover Structure of Management Information (SMI), textual conventions, and conformance statements, as well as various MIBs. These are covered in STD 16 and STD 17 documents along with SMIv2 documents [RFC 2578–2580].

Message Handling and PDU Handling sets of documents address transport mappings, message processing and dispatching, protocol operations, applications, and access control. These would correspond to the SNMP STD 15 documents and the draft documents on SNMPv2 [RFC 1905–1907] shown in Figure 4.9. RFCs [2573–2575] address these in SNMPv3.

7.3 ARCHITECTURE

An SNMP management network consists of several nodes, each with an SNMP entity. They interact with each other to monitor and manage the network and resources. The architecture of an SNMP entity

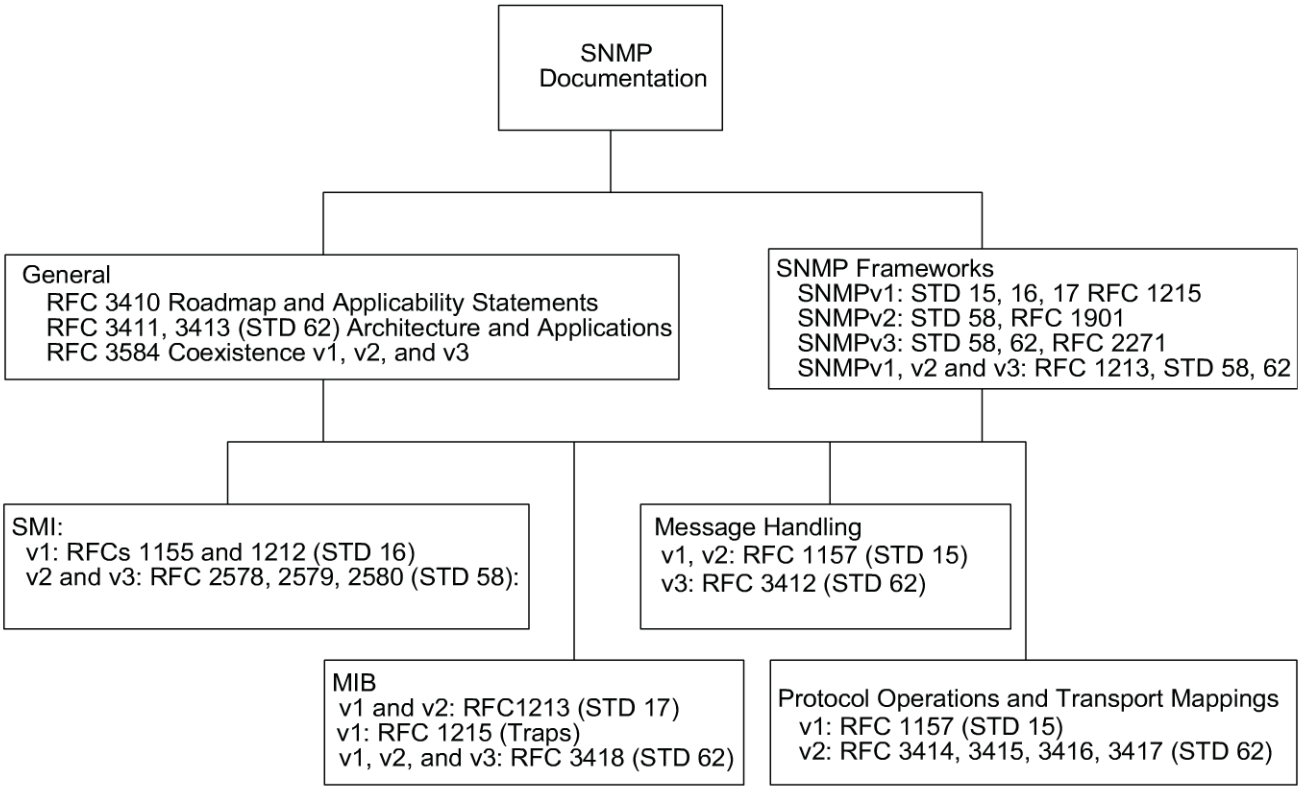

Figure 7.1 SNMP Documentation (Recommended in SNMPv3)

is defined as the elements of an entity and the names associated with them. There are three kinds of naming: naming of entities, naming of identities, and naming of management information. Let us first look at the elements of an entity, including its naming.

7.3.1 Elements of an Entity

The elements of the architecture associated with an SNMP entity, shown in Figure 7.2, comprise an SNMP engine and a set of applications. The SNMP engine, named *snmpEngineID*, comprises a dispatcher, message processing subsystem, security subsystem, and an access control subsystem.

SNMP Engine. As shown in Figure 7.2, an SNMP entity has one SNMP engine, which is uniquely identified by an *snmpEngineID*. The SNMP engine ID is made up of octet strings. The length of the ID is 12 octets for SNMPv1 and SNMPv2, and is variable for SNMPv3. This is shown in Figure 7.3. The first four octets in both formats are set to the binary equivalent of the agent's SNMP management private enterprise number. The first bit of the four octets is set to 1 for SNMPv3 and 0 for earlier versions. For example, if Acme Networks has been assigned {enterprises 696}, the first four octets would read '800002b8'H in SNMPv3 and '000002b8'H in SNMPv1 and SNMPv2.

The fifth octets for SNMPv1 and SNMPv2 indicate the method that the enterprise used for deriving the SNMP engine ID and 6–12 octets function of the method. For a simple entity, it could be just the IP address of the entity.

The fifth octet of the SNMPv3 engine ID indicates the format used in the rest of the variable number of octets. Table 7.2 shows the values of the fifth octet for SNMPv3.

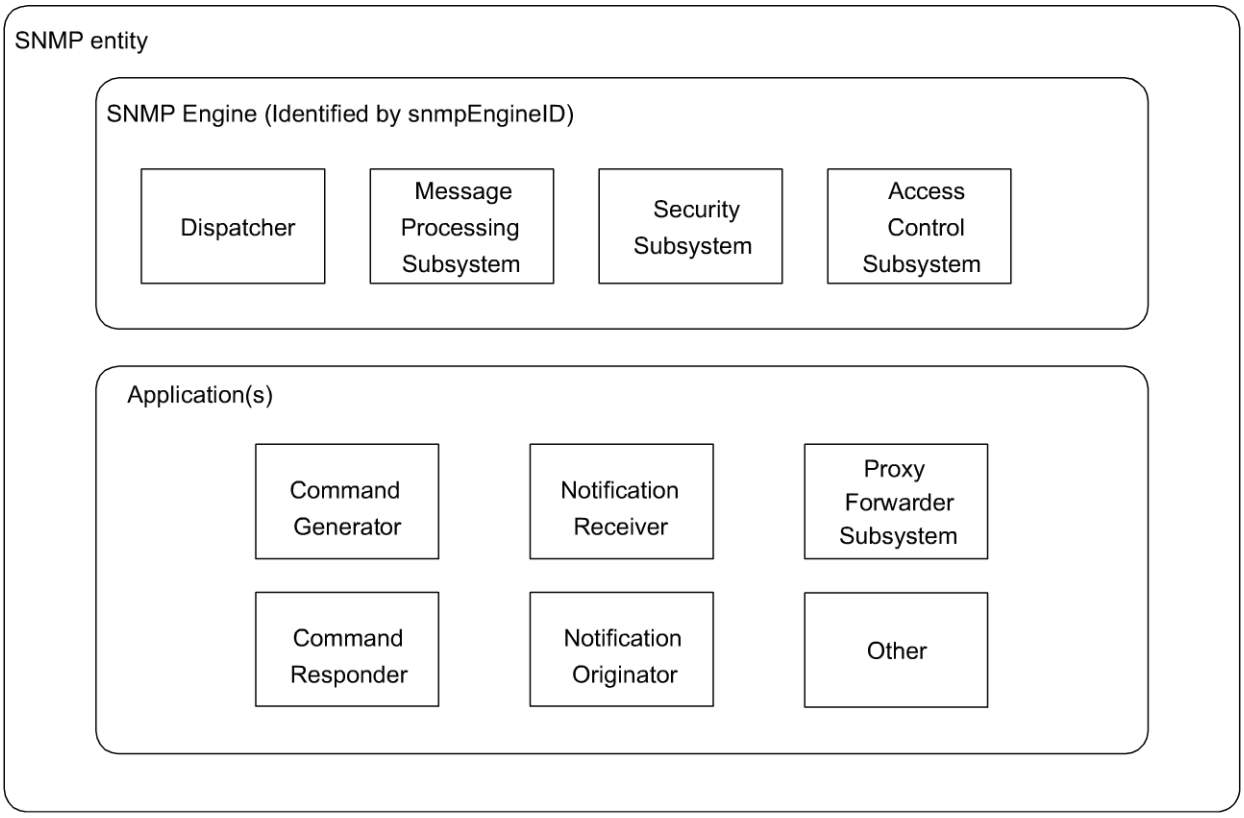

Figure 7.2 SNMPv3 Architecture

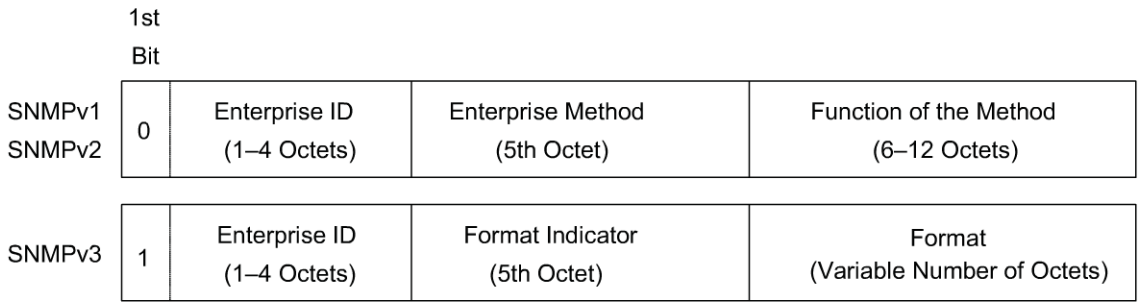

Figure 7.3 SNMP Engine ID

Dispatch Subsystem. There is only one dispatcher in an SNMP engine and it can handle multiple versions of SNMP messages. It does the following three sets of functions. First, it sends messages to and receives messages from the network. Second, it determines the version of the message and interacts with the corresponding MPM. Third, it provides an abstract interface (described in Section 7.3.3) to SNMP applications to deliver an incoming PDU to the local application and to send a PDU from the local application to a remote entity.

The three separate functions in the dispatcher subsystem are accomplished using: (1) a transport mapper; (2) a message dispatcher; and (3) a PDU dispatcher. The transport mapper delivers the message over the appropriate transport protocol of the network. The message dispatcher routes the outgoing and incoming messages to the appropriate module of the message processor. If a message is received for an SNMP version, which is not handled by the message processing subsystem, it would be rejected by the

Table 7.2 SNMPv3 Engine ID Format (5th Octet)

0	Reserved, unused
1	IPv4 address (4 octets)
2	IPv6 (16 octets)
	Lowest non-special IP address
3	MAC address (6 octets) Lowest IEEE MAC address, canonical order
4	Text, administratively assigned
	Maximum remaining length 27
5	Octets, administratively assigned
	Maximum remaining length 27
6–127	Reserved, unused
128–255	As defined by the enterprises Maximum remaining length 27

message dispatcher. The PDU dispatcher within an SNMP entity handles the traffic routing of PDUs between applications and the Message Processor Model.

Message Processing Subsystem. The SNMP message processing subsystem of an SNMP engine interacts with the dispatcher to handle version-specific SNMP messages. It contains one or more MPMs. The version is identified by the version field in the header.

Security and Access Control Subsystems. The security subsystem provides security services at the message level in terms of authentication and privacy protection. The access control subsystem provides access authorization service.

Applications Module. The application(s) module is made up of one or more applications, which comprise command generator, notification receiver, proxy forwarder, command responder, and notification originator. The first three applications are normally associated with an SNMP manager and the last two with an SNMP agent. The application(s) module may also include other applications, as indicated by the box, "Other," in Figure 7.2.

7.3.2 Names

Naming of entities, identities, and management information is part of SNMPv3 specifications. We already mentioned the naming of an entity by its SNMP engine ID, *snmpEngineID*. Two names are associated with identities, *principal* and *securityName*. *Principal* is the "who" requesting services. It could be a person or an application. The *securityName* is a human readable string representing a principal. The principal could be a single user; for example, name a network manager or a group of users, such as names of operators in the network operations center. It is made non-accessible. It is hidden and is based on the security model (SM) used. However, it is administratively given a security name; for example, User 1 or Admin, which is made readable by all.

A management entity can be responsible for more than one managed object. For example, a management agent associated with a managed object at a given node could be managing a neighboring node besides its own. Each object is termed *context* and has a *contextEngineID* and a *contextName*. When there is a one-to-one relationship between the management entity and the managed object, *contextEngineID* is the same as *snmpEngineID*. A *scopedPDU* is a block of data containing a *contextEngineID*, a *contextName*, and a PDU. An example of this would be a switched hub where a common SNMP agent in the hub is accessed to manage the interfaces of the hub. The agent would have an *snmpEngineID* and each interface card would have a context Engine ID. In contrast, in a non-switched hub with each interface card being managed individually, the snmpEngineID and contextID are the same.

7.3.3 Abstract Service Interfaces

The subsystems in an SNMP entity communicate with each other across an interface, a subsystem providing a service and the other using the service. We can define a service interface between the two. If the interface is defined such that it is generic and independent of specific implementation, it becomes a conceptual interface, termed *abstract service interface*. These abstract services are defined by a set of primitives that define the services. Figure 7.4(a) shows subsystem A sending a request for service using the primitive *primitiveAB* to subsystem B. The *primitiveAB* is associated with the receiving subsystem B, which is the one that is providing the service in this illustration. A primitive has IN and OUT as operands or parameters, which are data values. These are indicated by a1 and a2, and b1 and b2, respectively. The IN parameters are input values to the called subsystem from the subsystem calling for service. The OUT parameters are the responses expected from the called subsystem to the calling subsystem. The OUT parameters are sent unfilled in the message format by the calling system (remember Get-Request PDU?) and are returned filled (Get-Response) by the called subsystem. When the calling subsystem expects a response from the called subsystem, there are directed messages in both directions with a two-directional arrow coupling the two, as shown in Figure 7.4(a). In this case the primitive *primitiveAB* is only indicated in the forward direction. In addition to returning the OUT parameters, the called subsystem could also return a value associated with the result of the request in terms of *statusInformation* or result, as shown in Figure 7.4(a). Because of the execution of *primitiveAB*, subsystem B may initiate a request for service from another subsystem, subsystem C, using *primitiveBC* over the abstract service interface between subsystems B and C.

In general, except for dispatcher, primitives are associated with the receiving subsystem. Dispatcher primitives are used in receiving messages from and to the application modules, as well as registering and unregistering them, and in transmitting and receiving messages from the network.

Figure 7.4(b) shows the example of the application, command generator, sending a request *sendPdu* (destined for a remote entity) to the dispatcher. The dispatcher, after successful execution of the service requested and sending it on the network, returns to the application *sendPduHandle*. The *sendPduHandle* will be used by the command generator to correlate the response from the remote entity. There are no OUT parameters to be filled in this primitive except the status information. However, the command generator does expect the status information, hence the coupling arrow indicator in the figure. The dispatcher sends an error indicator instead of *sendPduHandle* for the status information if the *sendPdu* transaction is a failure. The dispatcher also generates a request to the MPM, *prepareOutgoingMessage*. The *prepareOutgoingMessage* has both IN and OUT parameters and hence information flows in both directions. The numerous IN and OUT parameters associated with primitives are not identified in the figure for the sake of simplicity.

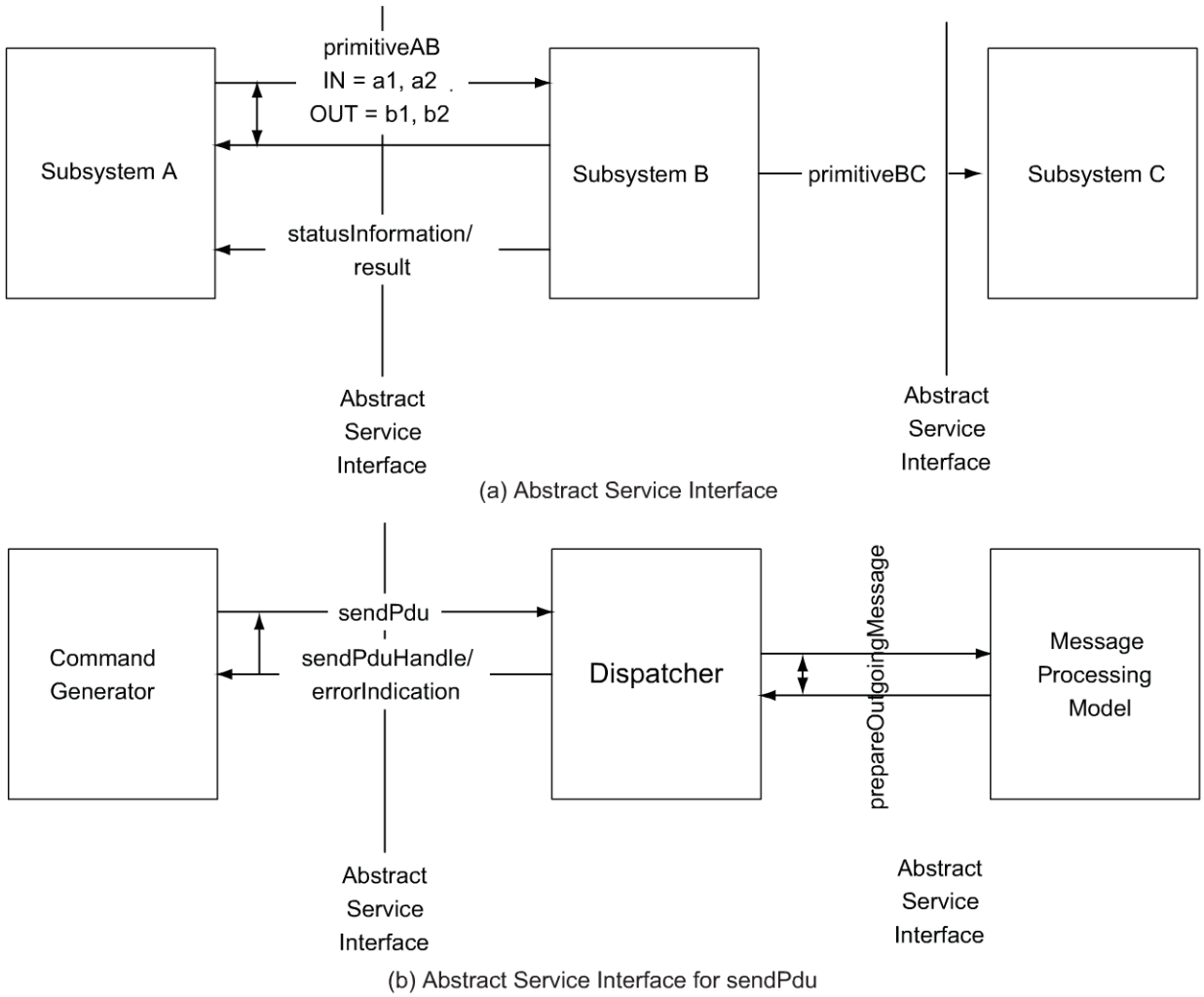

Figure 7.4 Abstract Service Interfaces

Table 7.3 lists the primitives served by the dispatcher, message processing subsystem, security, and access control subsystems. A brief description is presented for each primitive on the service provided and the user of service.

7.4 SNMPv3 APPLICATIONS

SNMPv3 formally defines five types of applications. These are not the same as the functional model that the OSI model addresses. These may be considered as the application service elements that are used to build applications. They are command generator, command responder, notification originator, notification receiver, and proxy forwarder. These are described in RFC 2273.

7.4.1 Command Generator

A command generator application is used to generate get-request, get-next-request, get-bulk, and set-request messages. The command generator also processes the response received for the command sent. Typically, the command generator application is associated with the network manager process.

Table 7.3 List of Primitives

MODULE	PRIMITIVE	SERVICE PROVIDED
Dispatcher	sendPdu	Processes request from application to send a PDU to a remote entity
Dispatcher	processPdu	Processes incoming message from a remote entity
Dispatcher	returnResponsePdu	Processes request from application to send a response PDU
Dispatcher	processResponsePdu	Processes incoming response from a remote entity
Dispatcher	registerContextEngineID	Registers request from a context engine
Dispatcher	unregisterContextEngineID	Unregisters request from a context engine
Message Processing Model	prepareOutgoingMessage	Processes request from dispatcher to prepare outgoing message to a remote entity
Message Processing Model	prepareResponseMessage	Processes request from dispatcher to prepare outgoing response to a remote entity
Message Processing Model	prepareDataElements	Processes request from dispatcher to extract data elements from an incoming message from a remote entity
Security Model	generateRequestMsg	Processes request from message processing model to generate a request message
Security Model	processIncomingMsg	Processes request from message processing model to process security data in an incoming message
Security Model	generateResponseMsg	Processes request from message processing model to generate a response message
Intra-Security Model	authenticateOutgoingMsg	Processes request to authentication service to authenticate outgoing message
Intra-Security Model	authenticateIncomingMsg	Processes request for authentication service to incoming message
Intra-Security Model	encryptData	Processes request from security model to privacy service to encrypt data
Intra-Security Model	decrypt Data	Processes request for privacy service to decrypt an incoming message
Access Control Model	isAccessAllowed	Processes request from application to access and authorize service requested

Figure 7.5 shows the use of the command generator application using the get-request example. In the top half of the figure, the get-request message is sent after it passes through the dispatcher, the MPM, and the SM. The command generator sends the *sendPdu* primitive to the dispatcher, which requests the

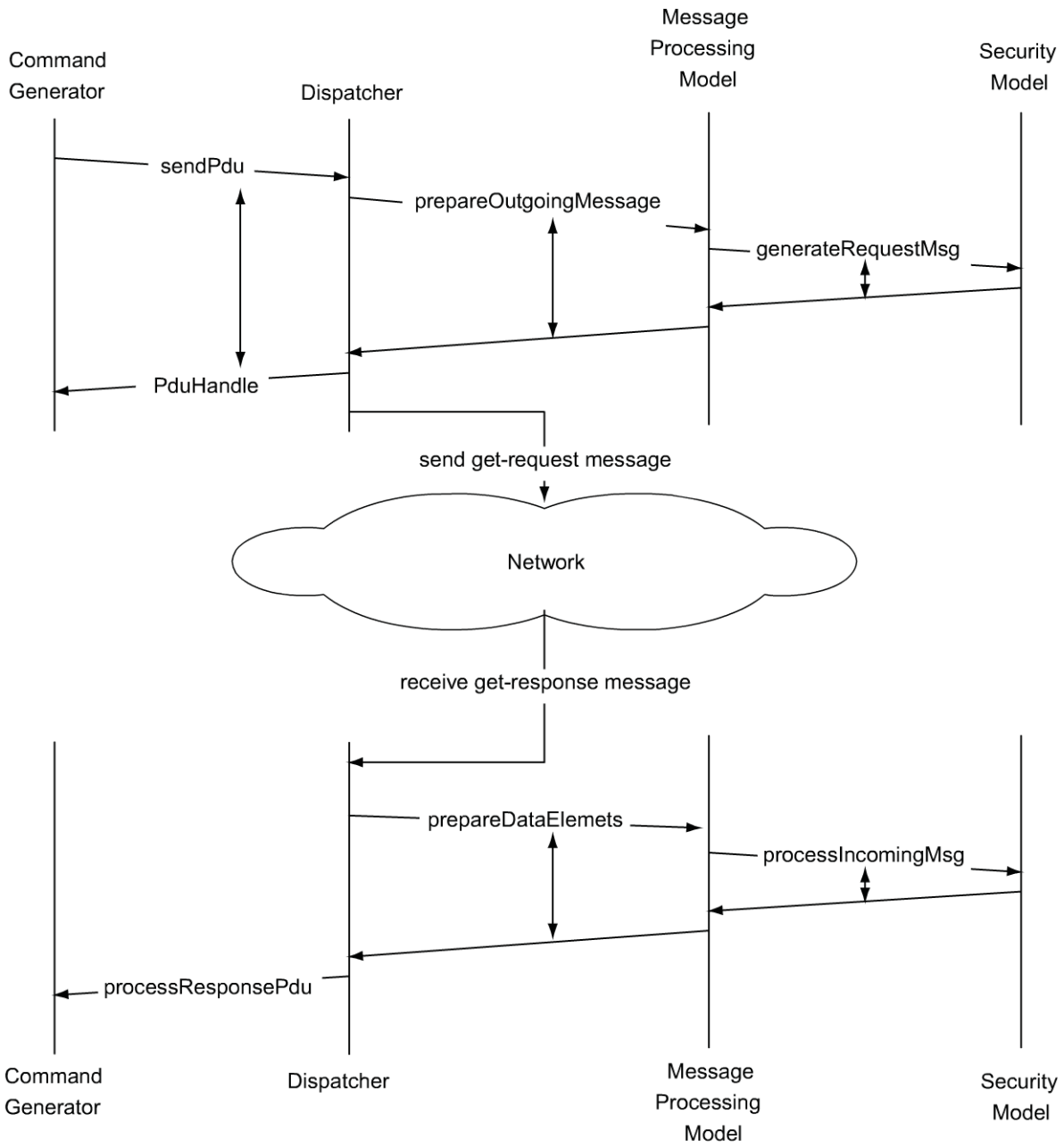

Figure 7.5 Command Generator Application

MPM to prepare an outgoing message. The dispatcher also sends a *sendPduHandle* to the command generator to track the request. The details on the information exchanged between MPM and SM are covered in Section 7.6. The SM is used to generate the outgoing message, including authentication and privacy parameters. The dispatcher then sends the message on the network.

The bottom half of Figure 7.5 presents the role of the command generator when the get-response message is received from the remote entity. The dispatcher receives the message from the network and requests MPM to prepare data elements, which are addressed in Section 7.6. The SM validates the authenticity and privacy parameters. The dispatcher receives the returned message from SM and forwards it to the command generator to process the response.

An example of the command generator transaction is an SNMPv1 get-request command from a network management system (NMS) to an agent requesting the values of System group (see Figure 5.17a). The command generator sends the command and OIDs to the dispatcher along with version number (SNMPv1) and security information. It also sends a tracking ID, *sendPduHandle*, to the NMS. This would be *Request ID* (=1) shown in Figure 5.17(a). When the MPM returns the outgoing message, which could be a secured (authenticated and encrypted) message, the dispatcher delivers it to the network using user datagram protocol (UDP) to be transmitted to the agent. The command generator receives the response from the dispatcher (asynchronously) sent by the agent. The primitive *processResponsePdu* would deliver the PDU containing the values for the System group shown in Figure 5.17(b) to the command generator. The command generator matches the response PDU received with *Request ID* = 1 with the one that was sent.

7.4.2 Command Responder

A command responder processes the get and set requests destined for it and is received from a legitimate non-authoritative remote entity. It performs the appropriate action of get or set on the network element, prepares a get-response message, and sends it to the remote entity that made the request. This is shown in Figure 7.6. In contrast to Figure 7.5, in which the top and bottom half processes run on two remote objects, the top and bottom of Figure 7.6 belong to the same object. Typically, the command responder is in the management agent associated with the managed object.

Before the get-request could be processed by the command-responder application, the context that the SNMP engine is responsible for must register with the SNMP engine. It does this by using the *registerContextEngineID*. Once this is in place, the get-request (same example as used in command generator) is received by the dispatcher, data elements are prepared by the MPM, security parameters are validated by the SM, and the *processPdu* is passed on to the command responder. This set of processes is presented in the top half of Figure 7.6.

Once the request is processed by the Command Responder, it prepares the get-response message as shown in the bottom half of Figure 7.6. The message is passed to the Dispatcher using the *returnResponsePdu*. The MPM prepares the response message, the SM performs the security functions, and the Dispatcher eventually transmits the get-response message on the network.

Continuing the example discussed in Section 7.4.1 for the command generator, the dispatcher in the SNMP agent receives the message. The message is processed by the MPM and the SM and is returned to the dispatcher. Assuming that the managed object has registered its context engine ID with the dispatcher using *registerContextEngineID*, the message is delivered to the command responder using *processPdu*. When the command responder acquires the System group information, it fills the PDU received with System group object values shown in Figure 5.17(b). The *returnResponsePdu* primitive is used by the command generator to deliver the message to the dispatcher. The dispatcher, after processing the get-response message through the MPM and the SM, transmits it across the network using UDP protocol.

7.4.3 Notification Originator

The notification originator application generates either a trap or an inform message. Its function is somewhat similar to the command responder, except that it needs to find out where to send the message and what SNMP version and security parameters to use. Further, the notification generator must determine the *contextEngineID* and context name of the context that has the information to be sent. It obtains

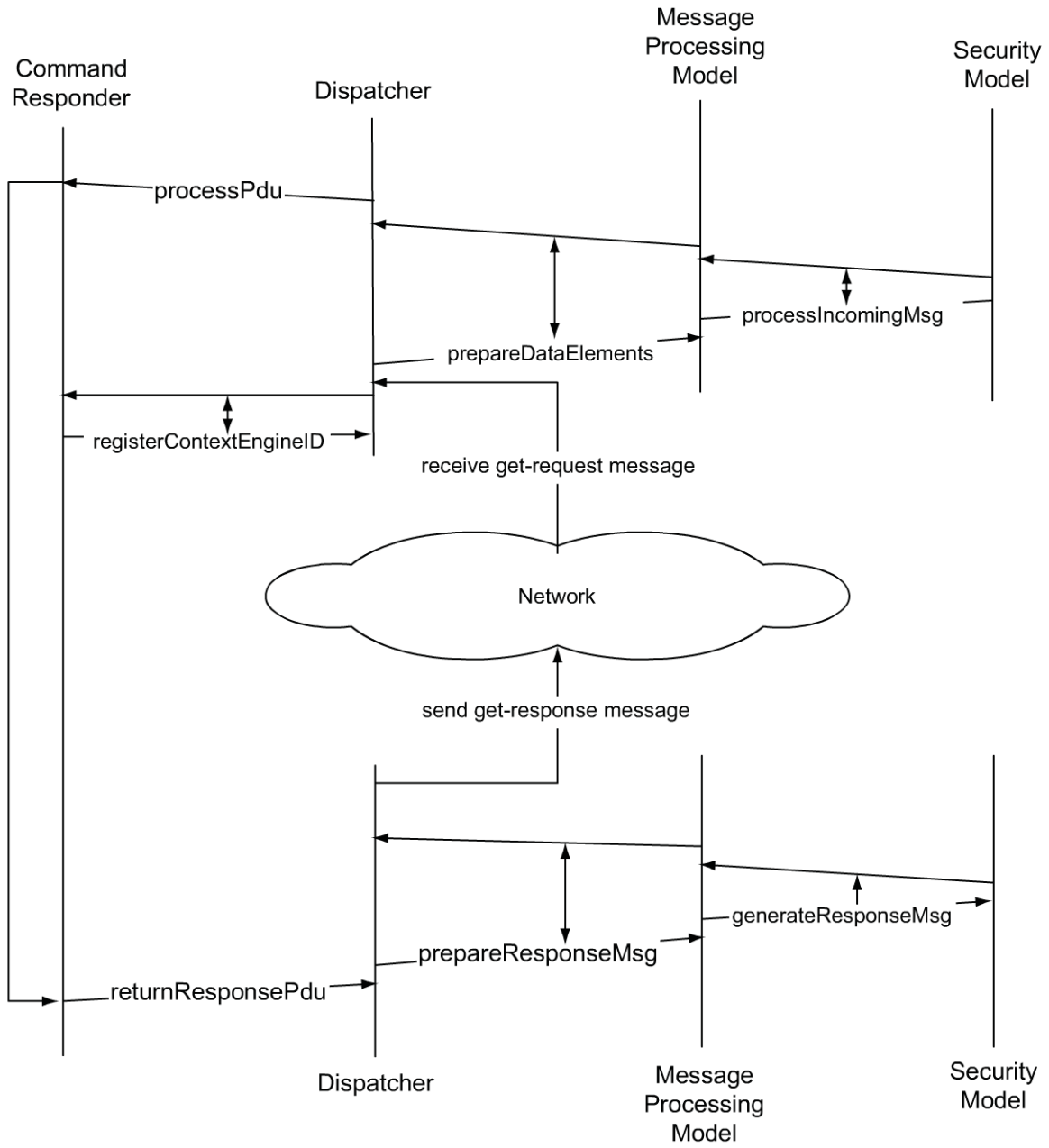

Figure 7.6 Command Responder Application

these data using newly created MIBs for the notification group and the target group, as well as using other modules in the system. We will learn about the new MIBs defining the new groups in Section 7.5. The notification group contains information on whether a notification should be sent to a target and, if so, what filtering should be used on the information. The target that the notification should be sent to is obtained from the target group.

7.4.4 Notification Receiver

The notification receiver application receives SNMP notification messages. It registers with the SNMP engine to receive these messages, just as the command responder does to receive get and set messages.

7.4.5 Proxy Forwarder

The proxy forwarder application performs a function similar to what we discussed in Chapter 6 on the proxy server. However, the proxy definition has been clearly defined and restricted in SNMPv3 specifications. The term "proxy" is used to refer to a proxy forwarder application that forwards SNMP requests, notifications, and responses without regard for what managed objects are contained in those messages. Non-SNMP object translation does not fall under this category. The proxy forwarder handles four types of messages: messages generated by the command generator, command responder, notification generator, and those that contain a report indicator. The proxy forwarder uses the translation table in the proxy group MIB created for this purpose.

7.5 SNMPv3 MANAGEMENT INFORMATION BASE

The new objects defined in SNMPv3 follow the textual convention specified in SNMPv2 and described in Section 6.3. Refer to the RFCs listed in Table 7.1 for complete details on managed objects and MIBs in SNMPv3. We will address a subset of the MIBs here. Figure 7.7 shows the MIB of the new object groups. They are nodes under *snmpModules* {1.3.6.1.6.3}, shown in Figure 6.31. There are seven new MIB groups. The *snmpFrameworkMIB*, node 10 under *snmpModules*, describes the SNMP Management architecture. The MIB group *snmpMPDMIB* (node 11) identifies objects in message processing and dispatching subsystems.

There are three groups defined under *snmpModules* for applications. They are *snmpTargetMIB* (node 12), *snmpNotificationMIB* (node 13), and *snmpProxyMIB* (node 14). The first two are used for notification generator. The *snmpTargetMIB* defines MIB objects, which are used to remotely configure the parameters used by a remote SNMP entity. There are two tables in that MIB, which are of specific interest for us. They are shown in Figure 7.8. The *snmpTargetAddrTable*, which is in *snmpTargetObjects* group, contains the addresses to be used in the generation of SNMP messages. There are nine columnar objects in the table, which are listed in Table 7.4.

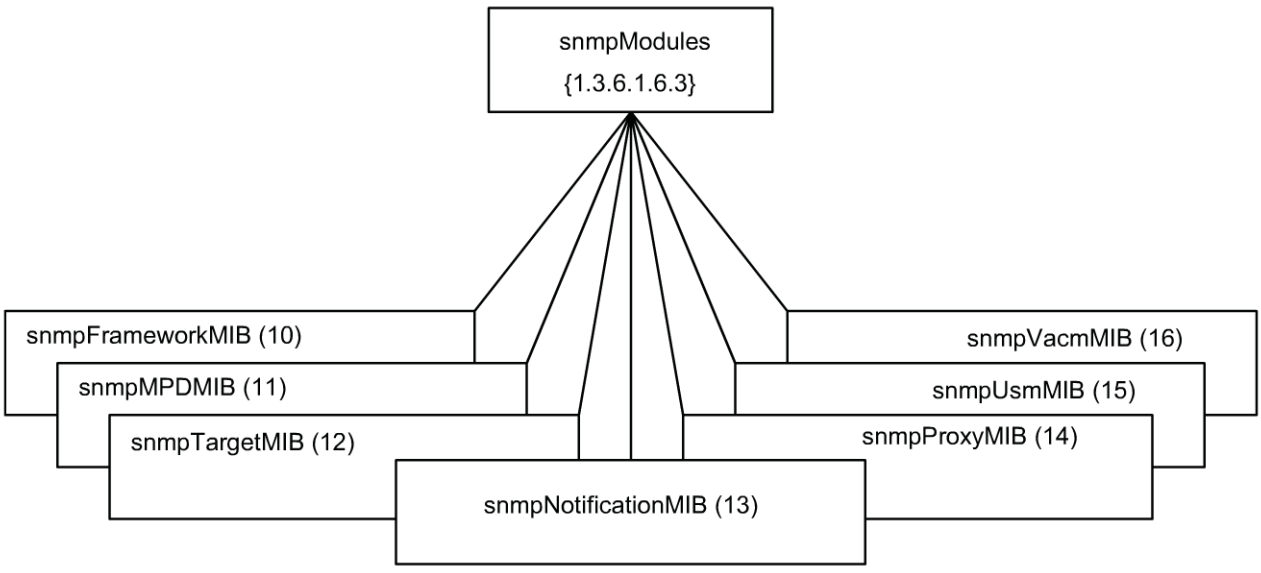

Figure 7.7 SNMPv3 MIB

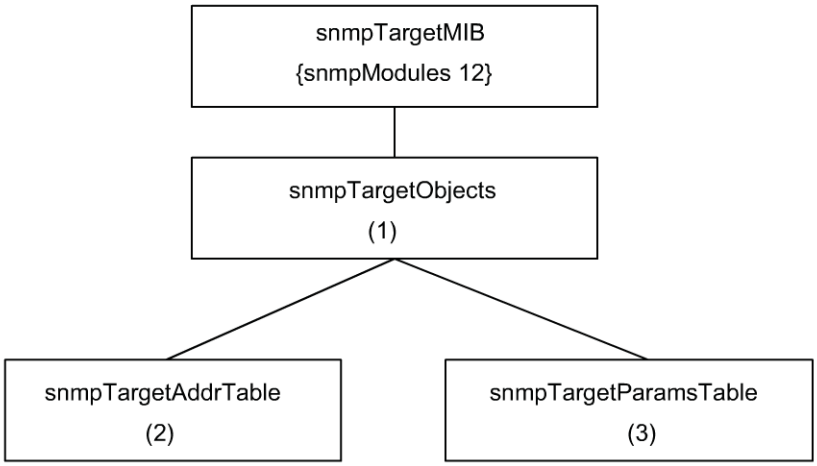

Figure 7.8 Target Address and Target Parameter Tables

Table 7.4 SNMP Target Address Table

ENTITY	OID	DESCRIPTION (BRIEF)
snmpTargetAddrTable	snmpTargetObjects 2	Table of transport addresses
snmpTargetAddrEntry	snmpTargetAddrTable 1	Row in the target address table
snmpTargetAddrName	snmpTargetAddrEntry 1	Locally administered name associated with this entry
snmpTargetAddrTDomain	snmpTargetAddrEntry 2	Transport type of the addresses
snmpTargetAddrTAddress	snmpTargetAddrEntry 3	Transport address
snmpTargetAddrTimeOut	snmpTargetAddrEntry 4	Reflects the expected maximum round-trip time
snmpTargetAddrRetryCount	snmpTargetAddrEntry 5	Number of retries
snmpTargetAddrTagList	snmpTargetAddrEntry 6	List of tag values used to select the target addresses for a particular operation
snmpTargetAddrParams	snmpTargetAddrEntry 7	Value that identifies an entry in the snmpTargetParams Table
snmpTargetAddrStorageType	snmpTargetAddrEntry 8	Storage type for this row
snmpTargetAddrRowStatus	snmpTargetAddrEntry 9	Status of the row

The second table in the *snmpTargetObjects* group is the *snmpTargetParamsTable*. The lead into this table is by using the columnar object *snmpTargetAddrParams* in the *snmpTargetAddrTable*. This contains the security parameters on authentication and privacy. The columnar objects in *snmpTargetParamsTable* are listed in Table 7.5.

The *snmpNotificationMIB* shown in Figure 7.9 deals with MIB objects for the generation of notifications. There are three tables in this group—namely, the notification table, the notification filter profile table, and the notification filter table. They are under the node *snmpNotifyObjects*. The SNMP notification table, *snmpNotifyTable*, is used to select management targets that should receive notifications, as well as the type of notification to be sent. Table 7.6 shows the columnar objects in the group.

The notification profile table group, *snmpNotifyProfileTable*, is used to associate a notification filter profile with a particular set of target parameters. The third group, the notification filter table,

Table 7.5 SNMP Target Parameters Table

ENTITY	OID	DESCRIPTION (BRIEF)
snmpTargetParamsTable	snmpTargetObjects 3	Table of SNMP target information to be used
snmpTargetParamsEntry	snmpTargetParamsTable 1	A set of SNMP target information
snmpTargetParamsName	snmpTargetParamsEntry 1	Locally administered name associated with this entry
snmpTargetParamsMPModel	snmpTargetParamsEntry 2	Message processing model to be used
snmpTargetParamsSecurityModel	snmpTargetParamsEntry 3	Security model to be used
snmpTargetParamsSecurityName	snmpTargetParamsEntry 4	Security name of the principal
snmpTargetParamsSecurityLevel	snmpTargetParamsEntry 5	Level of security
snmpTargetParamsStorageType	snmpTargetParamsEntry 6	Storage type for the row
snmpTargetParamsRowStatus	snmpTargetParamsEntry 7	Status of the row

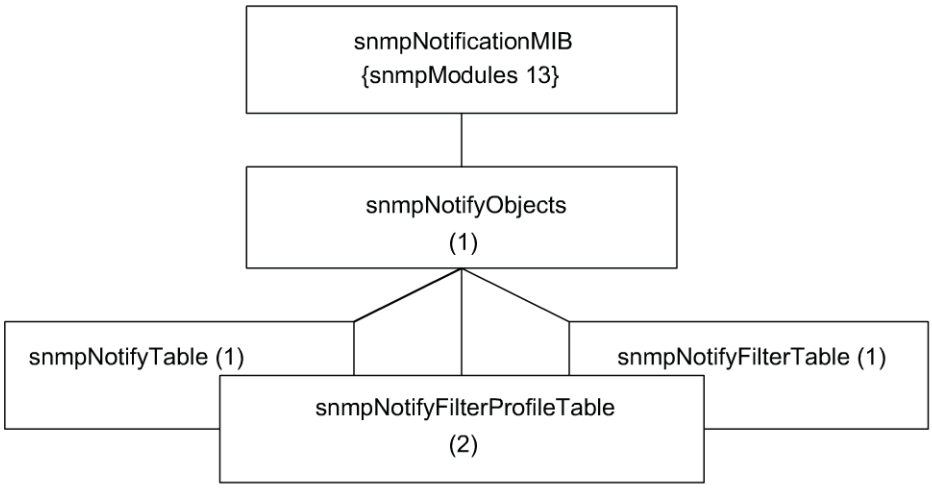

Figure 7.9 SNMP Notification Tables

Table 7.6 SNMP Notification Table

ENTITY	OID	DESCRIPTION (BRIEF)
snmpNotifyTable	snmpNotifyObjects 1	List of targets and notification types
snmpNotifyEntry	snmpNotifyTable 1	Set of management targets and the type of notification
snmpNotifyName	snmpNotifyEntry 1	Locally administered name associated with this entry
snmpNotifyTag	snmpNotifyEntry 2	A single value that is used to select entries in the snmpTargetAddrTable
snmpNotifyType	snmpNotifyEntry 3	Selects trap or inform to send
snmpNotifyStorageType	snmpNotifyEntry 4	Storage type for the row
snmpNotifyRowStatus	snmpNotifyEntry 5	Status of the row

snmpNotifyFilterTable, contains table profiles of the targets. The profile specifies whether a particular target should receive particular information.

The *snmpProxyMIB* is concerned with objects in a proxy forwarding application, such as the SNMPv2 proxy server shown in Figure 6.46. It contains a table of translation parameters used by the proxy forwarder application for forwarding SNMP messages.

The SNMP USM objects are defined in *snmpUsmMIB* module (node 15). Lastly, the objects for VACM for SNMP are defined in the *snmpVacmMIB* module (node 16). We will discuss the details of these MIBs later when we address security in the next section and access control in Section 7.8.

7.6 SECURITY

One of the main objectives, if not the main objective, in developing SNMPv3 is the addition of security features to SNMP management. Authentication and privacy of information, as well as authorization and access controls, have been addressed in SNMPv3 specifications. We will cover the authentication and privacy issues in this section and in Section 7.7. We will deal with access control in Section 7.8.

SNMPv3 architecture permits flexibility to use any protocol for authentication and privacy of information. However, the IETF SNMPv3 working group has specified a USM for its security subsystem. It is termed user-based as it follows the traditional concept of a user, identified by a user name with which to associate security information. The working group has specified HMAC-MD5-96 and HMAC-SHA-96 (see Section 7.7.1 for an explanation) as the authentication protocols. Cipher Block Chaining mode of Data Encryption Standard (CBC-DES) has been adopted for privacy protocol.

We will discuss the general aspects of security associated with the types of threats, the security modules, the message data format to accommodate security parameters, and the use and management of keys in this section. We will specifically address the USM in the next section.

7.6.1 Security Threats

Four types of threats exist to network management information while it is being transported from one management entity to another: (1) modification of information, (2) masquerade, (3) message stream modification, and (4) disclosure. These are shown in Figure 7.10, where the information is transported from management entity A to management entity B. For the first three threats, the signal has to be intercepted by an intruder to tamper with it, whereas for the disclosure threat, the signal can just be tapped and not intercepted.

Modification of information is the threat that some unauthorized user may modify the contents of the message while it is in transit. Data contents are modified, including falsifying the value of an object. It does not include changing the originating or destination address. The modified message is received by entity B, which is unaware that it has been modified. For example, the response by an SNMP agent to a request by an SNMP manager could be altered by this threat.

Masquerade is when an unauthorized user sends information to another assuming the identity of an authorized user. This can be done by changing the originating address. Using the masquerade and modification of information, an unauthorized user can perform operation on a management entity, which he or she is not permitted to do. The SNMP set operation should be protected against this attack.

The SNMP communication uses connectionless transport service, such as UDP. This means that the message could be fragmented into packets with each packet taking a different path. The packets could

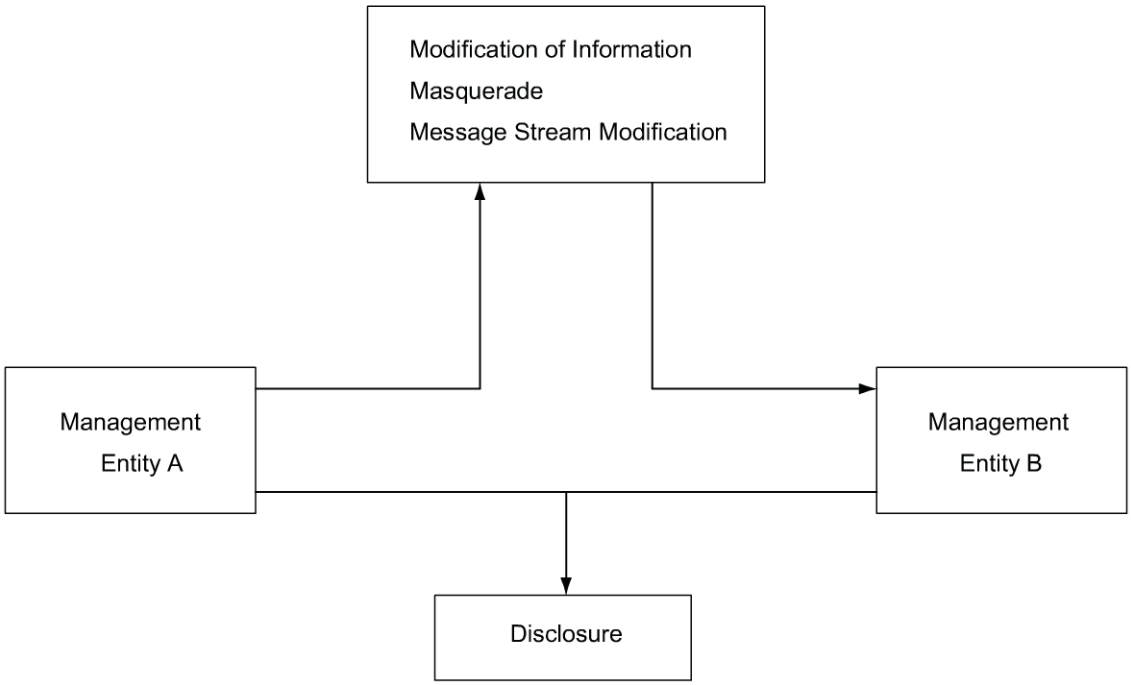

Figure 7.10 Security Threats to Management Information

arrive at the destination out of sequence and have to be reordered. The threat here is that the intruder may manipulate the message stream (**message stream modification**) and maliciously reorder the data packets to change the meaning of the message. For example, the sequence of data of a table could be reordered to change the values in the table. The intruder could also delay messages so that those messages arrive out of sequence. The message could be interrupted, stored, and replayed at a later time by an unauthorized user.

The fourth and last threat that is shown in Figure 7.10 is **disclosure** of management information. The message need not be intercepted for this, but just eavesdropped. For example, the message stream of accounting could be promiscuously monitored by an employee with a TCP/IP dump procedure, and then the information could be used against the establishment.

There are at least two more threats that would be considered as threats in traditional data communication, but the SNMP SM has classified them as being non-threats. The first is denial of service, when an authorized user is denied service by a management entity. This is considered as not being a threat, as a network failure could cause such a denial. It is the responsibility of the protocol to address this issue. The second threat that is not considered a management threat is traffic analysis by an unauthorized user. It was determined by the IETF SNMPv3 working group that there is no significant advantage achieved by protecting against this attack.

7.6.2 Security Model

In normal operational procedures, the MPM in the message processing subsystem interacts with security subsystem models. For example, in Figure 7.2, an outgoing message is generated by an application, which is first handled by a dispatcher subsystem, then by an MPM, and finally by the SM. If the message is to be authenticated, the SM authenticates it and forwards it to the MPM. Similarly for an incoming message, the MPM requests the service of the security subsystem to authenticate the user ID.

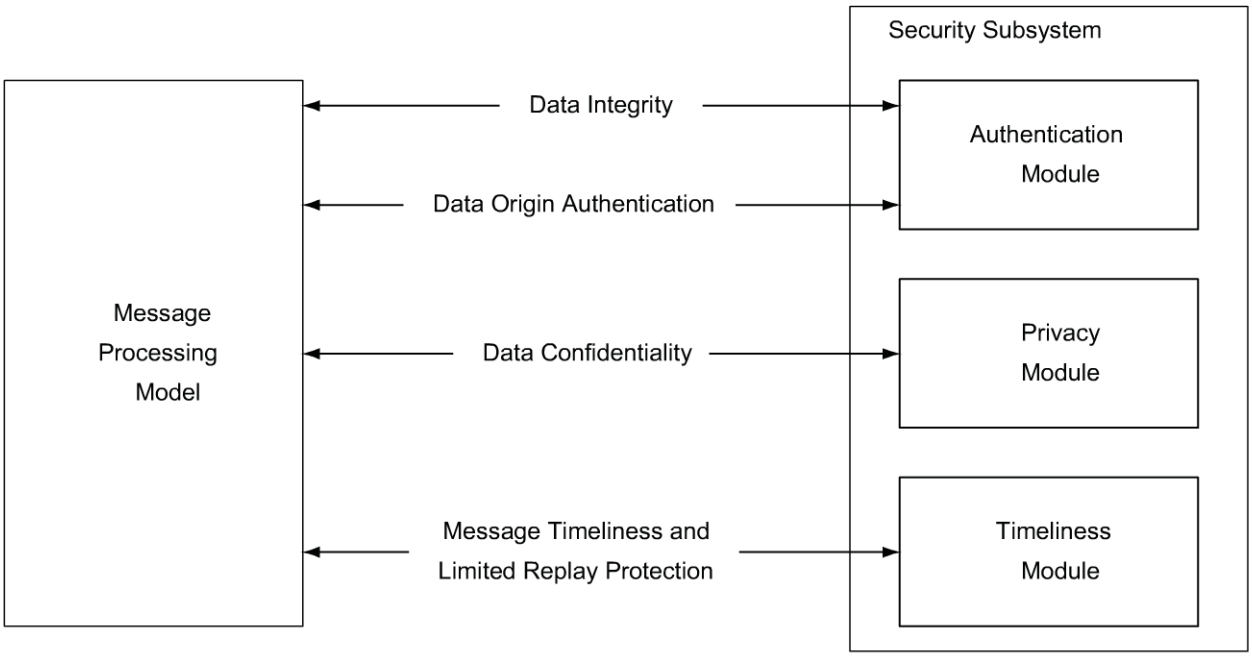

Figure 7.11 Security Services

Figure 7.11 shows the services provided by the three modules in the security subsystem to the MPM. They are the authentication module, the privacy module, and the timeliness module.

Authoritative SNMP Engine. When two management entities communicate, the services provided by each are determined by the role they play, i.e., whether the entity is authorized to perform the service. This led to the concept of authoritative and non-authoritative SNMP engines. This is dependent on which SNMP engine controls the communication between the two entities. SNMPv3 architecture defines that in a communication between two SNMP engines, one acts as an *authoritative engine* and the other as a *non-authoritative engine*. There is a well-defined set of rules as to who is the authoritative SNMP engine for each message that is communicated between two SNMP engines. For get-request, get-next-request, get-bulk-request, set-request or inform messages, the receiver of the message is the authoritative SNMP engine. Since these messages are originated by a manager process in a network management system (NMS), the receiver is the SNMP agent. Thus, the agent is the authoritative SNMP engine. For trap, get-response, and report messages, the sender or the agent is the authoritative SNMP engine. Thus, an SNMP engine that acts in the role of an agent is the designated authoritative SNMP engine. The SNMP engine that acts in the role of a manager is the non-authoritative engine. In general, an SNMP agent is the authoritative SNMP engine in SNMP communication.

An authoritative SNMP engine is responsible for the accuracy of the time-stamp and a unique SNMP engine ID in each message. This requires that every non-authoritative SNMP engine keep a table of the time and authoritative engine ID of every SNMP engine that it communicates with.

Security Authentication. Communication between two entities could satisfy the condition of authoritative and non-authoritative pair. However, it should be the right set of pairs. Thus, the source from which the message is received should be authenticated by the receiver. Further, authentication is needed for the security reasons discussed in Section 7.6.1. Security authentication is done by the authentication module in the security subsystem.

The *authentication module* provides two services, *data integrity* and *data origin authentication*. The data integrity service provides the function of authenticating a message at the originating end and validating it at the receiving end, ensuring that it has not been modified in the communication process by an unauthorized intruder. Authentication validation also catches any non-malicious modification of data in the communication channel. The authentication scheme uses authentication protocols, such as HMAC-MD5-96 or HMAC-SHA-96 in SNMPv3 or any other protocol in place of it.

The second service that is provided by the authentication module is *data origin authentication*. This ensures that the claimed identity of the user on whose behalf the message was sent is truly the originator of the message. The authentication module appends to each message a unique identifier associated with an authoritative SNMP engine.

Privacy of Information. The second module in the security subsystem in Figure 7.11 is the *privacy module*, which provides *data confidentiality service*. Data confidentiality ensures that information is not made available or disclosed to unauthorized users, entities, or processes. The privacy of the message is accomplished by encrypting the message at the sending end and decrypting it at the receiving end.

Timeliness of Message. The *timeliness module* is the third module in the security subsystem and provides the function of checking *message timeliness* and thus prevents message redirection, delay, and replay. Using the concept of an authoritative SNMP engine, a window of time is set in the receiver to accept a message. The travel time between the sender and the receiver should be within this time window interval. The time clock in both the sender and the receiver is synchronized to the authoritative SNMP engine. The recommended value for the window time in SNMPv3 is 150 seconds.

For implementation of the timeliness module, the SNMP engine maintains three objects: *snmpEngineID, snmpEngineBoots,* and *snmpEngineTime*. The *snmpEngineID* uniquely identifies the authoritative SNMP engine. The *snmpEngineBoots* is a count of the number of times the SNMP engine has re-booted or re-initialized since *snmpEngineID* was last configured. The *snmpEngineTime* is the number of seconds since the *snmpEngineBoots* counter was last initialized or reset.

The timeliness module also checks the message ID of a response with the request message and drops the message if they do not match.

We will next look at the message format in SNMPv3 in general, and the security parameters contained in it in particular.

7.6.3 Message Format

The SNMPv3 message format is shown in Figure 7.12. It consists of four groups of data. Details of the fields in each group except security parameters are given in Table 7.7. The first group is a single field, which is the version number and is in the same position as in SNMPv1 and SNMPv2.

Global (header) data defined by the data type header are the second group of data in the message format. They contain administrative parameters of the message, which are message ID, message maximum size, message flag, and message SM. It is worth noting that an SNMP engine can handle many models concurrently in the message processing subsystem. The dispatcher subsystem examines the version number in the message and sends it to the appropriate message processing module in the message subsystem. For example, if the version is set to snmpv2, the SNMPv2 message processing module would be invoked.

The third group of data, security parameter fields, are used by the SM in communicating between sending and receiving entities. The values of the parameters depend on the message SM set in the header data. The parameters are shown in Figure 7.12 and will be discussed in Section 7.7.

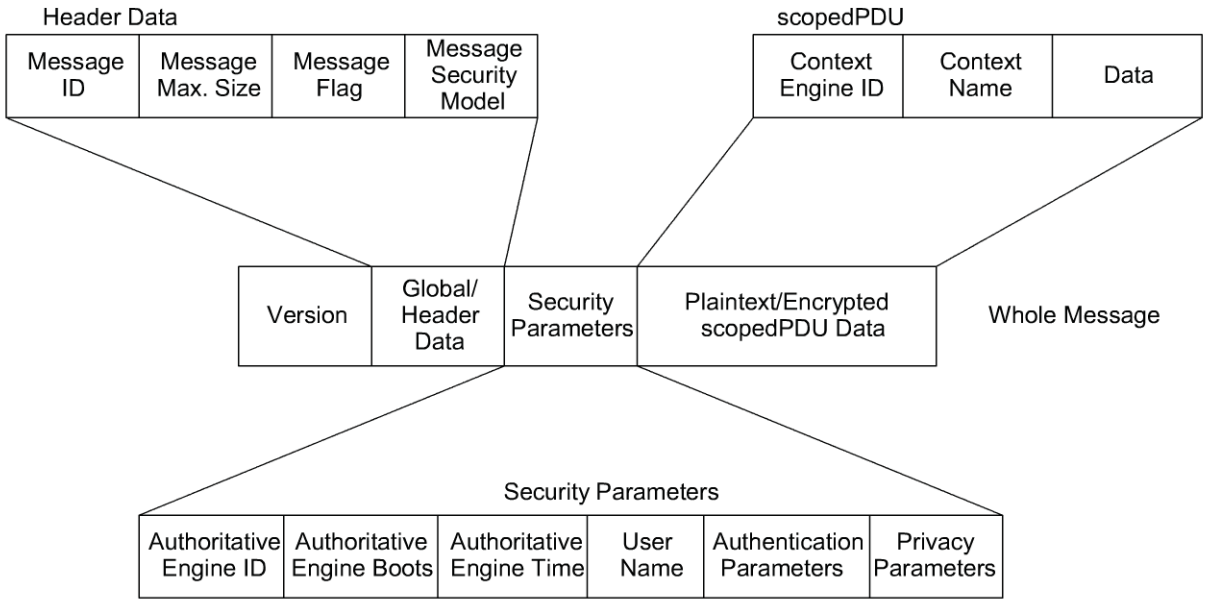

Figure 7.12 SNMPv3 Message Format

Table 7.7 SNMPv3 Message Format

FIELD	OBJECT NAME	DESCRIPTION
Version	msgVersion	SNMP version number of the message format
Message ID	msgID	Administrative ID associated with the message
Message max. size	msgMaxSize	Maximum size supported by the sender
Message flags	msgFlags	Bit fields identifying report, authentication; and privacy of the message
Message security model	msgSecurityModel	Security model used for the message; concurrent multiple models allowed
Security parameters (See Table 7.8)	msgSecurityParameters	Security parameters used for communication between sending and receiving security modules
Plaintext/encrypted scopedPDU data	scopedPduData	Choice of plaintext or encrypted scopedPDU; scopedPDU uniquely identifies context and PDU
Context engine ID	contextEngineID	Unique ID of a context (managed entity) with a context name realized by an SNMP entity
Context name	contextName	Name of the context (managed entity)
PDU	data	Contains unencrypted PDU

The fourth and final group of fields in the whole message record shown in Figure 7.12 is the plaintext/encrypted *scopedPDU* data. The *scopedPduData* field contains either unencrypted or encrypted *scopedPDU*. If the privacy flag is set to zero (no privacy) in the message flag (see header data), then this field contains plaintext *scopedPDU*, which is unencrypted *scopedPDU*. The plaintext *scopedPDU*

comprises the context engine ID, context name, and the PDU. A management entity can be responsible for multiple instances of managed objects. For example, in an ATM switch, a single managed entity acts as the agent for all the network interface cards in all its ports. We could treat each interface card as a context with a context engine ID and a context name. Thus, a particular context name, in conjunction with a particular context engine ID, identifies the particular context associated with the management information contained in the PDU portion of the message. The object name for PDU is *data*.

7.7 SNMPv3 USER-BASED SECURITY MODEL

The security subsystem for SNMPv3 is a USM, which is based on the traditional user name concept. Just as we have defined abstract service interfaces between various subsystems in an SNMP entity, we can define abstract service interfaces in USM. They define conceptual interfaces between generic USM services and self-contained authentication and privacy services. There are two primitives associated with authentication service, one to generate an outgoing authenticated message (*authenticateOutgoingMsg*) and another to validate the authenticated incoming message (*authenticateIncomingMsg*). Similarly, there are two primitives associated with privacy services, *encryptData* and *decryptData*, for the encryption of outgoing messages and the decryption of incoming messages. These were included in the list of primitives in Table 7.3.

Services provided by authentication and privacy modules in the security subsystem for outgoing and incoming messages are shown in Figures 7.13 and 7.14, respectively. Looking at the overall picture, the MPM invokes the USM in the security subsystem. The USM in turn invokes, based on the security level set in the message, the authentication and privacy modules. Results are returned to the MPM by the USM.

In Figure 7.13 that shows the process of an outgoing message, we will assume that both privacy and authentication flags are set in the message flag in the header data. The MPM inputs the MPM information, header data, security data, and *scopedPDU* to the security subsystem. The USM invokes the privacy module first, providing the encryption key and *scopedPDU* as input. The privacy module outputs

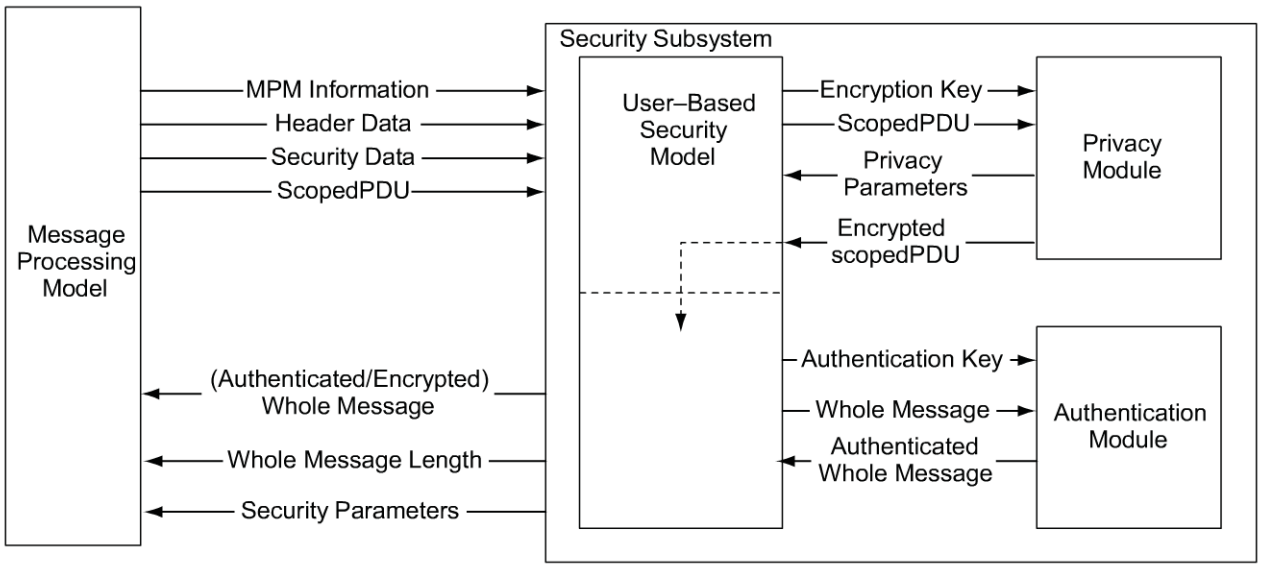

Figure 7.13 Privacy and Authentication Service for an Outgoing Message

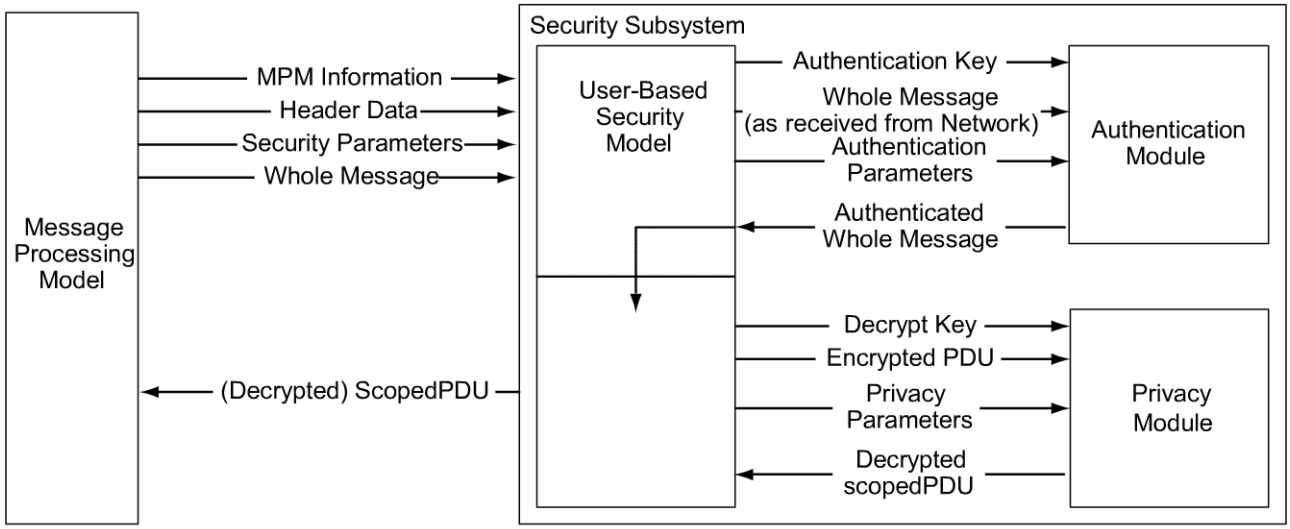

Figure 7.14 Privacy and Authentication Service for an Incoming Message

privacy parameters that are sent as part of the message and the encrypted *scopedPDU*. The USM passes the unauthenticated whole message with encrypted *scopedPDU* to the authentication module along with the authentication key. The authentication module returns the authenticated whole message to USM. The security subsystem returns the authenticated and encrypted whole message along with the message length and security parameters to the MPM.

Figure 7.14 shows the reverse process of an incoming message going through the authentication validation first, and then decryption of the message by the privacy module.

The security parameters used in the SM are shown in Figure 7.12. Table 7.8 lists the parameters and the corresponding SNMPv3 MIB objects. The position of the relevant MIB objects associated with the security parameters belongs to the two modules, *snmpFramesworkMIB* and *snmpUsmMIB*, under *snmpModulesMIB* shown in Figure 7.7. The details of the position of the objects in the MIB are presented in Figure 7.15.

We have already discussed the first three parameters in Table 7.8 associated with engine ID, number of boots, and time since the last boot. They are in the *snmpEngine* group shown in Figure 7.15. The last three parameters in the table are in *usmUserTable* in the *usmUser* group shown in Figure 7.15.

The fourth parameter is the user (principal) on whose behalf the message is being exchanged. The authentication parameters are defined by the authentication protocol columnar object in the *usmUserTable*. The *usmUserTable* describes the users configured in the SNMP engine and the authentication parameters the type of authentication protocol used. Likewise, the privacy parameters describe the type of privacy protocol used.

The *usmUserSpinLock* is an advisory lock that is used by SNMP command generator applications to coordinate their use of the set operation in creating or modifying secrets in *usmUserTable*.

Now that we have a broad picture, let us return to Figure 7.13 and follow through the detailed data flow and processes involved in the USM. Figure 7.13 shows the operation for an outgoing message, which could be either a Request message or a Response message. The MPM inputs information on the *message processing model* to be used (normally SNMP version number), leader data, security data (SM, SNMP engine ID, security name, and security level) and *scopedPDU* to the Security Subsystem (SS). This information is received by the User-base Security Model (UCM) in SS.

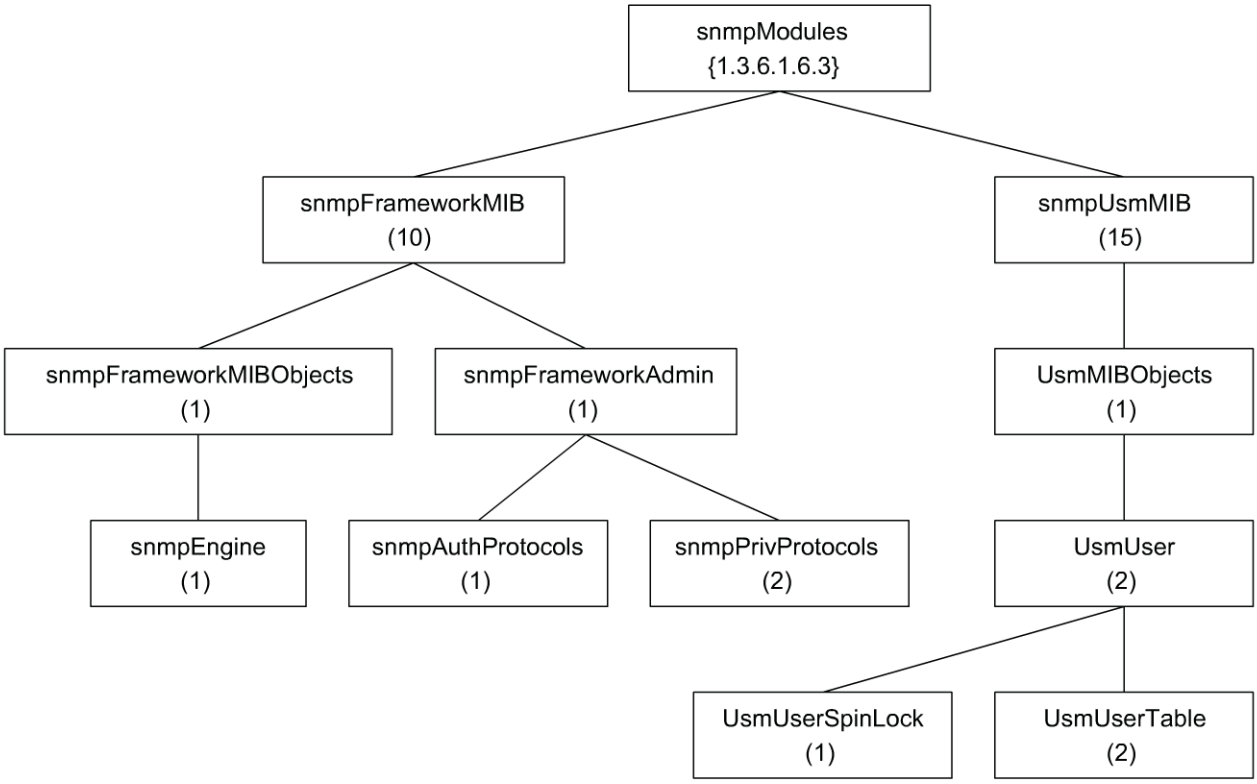

Figure 7.15 SNMPv3 MIB Objects for Security Parameters

Table 7.8 Security Parameters and Corresponding MIB Objects

SECURITY PARAMETERS	USM USER GROUP OBJECTS
msgAuthoritativeEngineID	snmpEngineID (under snmpEngine Group)
msgAuthoritativeEngineBoots	snmpEngineBoots (under snmpEngine Group)
msgAuthoritativeEngineTime	snmpEngineTime (under snmpEngine Group)
msgUserName	usmUserName (in usmUserTable)
msgAuthenticationParameters	usmUserAuthProtocol (in usmUserTable)
msgPrivacyParameters	usmUserPrivProtocol (in usmUserTable)

In the USM, the security-level settings for privacy and authentication determine the modules invoked. The encryption key and *scopedPDU* (context engine ID, context name, and PDU) are fed into the privacy module, which encrypts the PDU and returns the encrypted PDU along with privacy parameters to the calling module, USM.

The USM then communicates with the authentication module. The USM inputs the encrypted whole message along with the authentication key. The authentication module returns the authenticated whole message to USM. The USM passes the authenticated and encrypted whole message, whole message length, and securities parameters back to the MPM.

The operation for an incoming message is shown in Figure 7.14. Inputs to the security subsystem are the MPM information, header data, security parameters for the received message, and the whole message. The output of the security subsystem is *scopedPDU* in plaintext format.

Within the security subsystem, the operational sequence of authentication and privacy for an incoming message are reversed from that of the outgoing message. The message is first sent to the authentication module with the authentication key, the whole message received from the network, and authentication parameters received from the network as inputs. It outputs an authenticated whole message to the calling module in USM. The USM then feeds the decrypt key, privacy parameters, and the encrypted *scopedPDU* and receives in return the decrypted *scopedPDU*. The decrypted *scopedPDU* is then passed on to the message processing module.

7.7.1 Authentication Protocols

The secret to security using the authentication and privacy schemes is the secret key that is shared between the sender and the user. There is a secret key for authentication and a secret key for encryption and decryption. The secret key for the User-based Security Module (USM) is developed from the user password. Two algorithms are recommended in SNMPv3 for developing the key from the password. They are HMAC-MD5-96 and HMAC-SHA-96. The first letter in the designation stands for the cryptographic hash function (H) used for generating the Message Access Code (MAC). The second part in the designation is the hashing algorithm used, the first one being the MD5 hashing algorithm, and the second one the SHA-1 hashing algorithm to generate MAC. The MAC is derived by truncating the hashing code generated to 96 bits as indicated by the last set of characters in the designation.

Authentication Key. The authentication key, the secret key for authentication, is derived from a chosen password of the user. The user in our case is the non-authoritative SNMP engine, which is generally an NMS. In both MD5 and SHA-1 algorithms, the password is repeated until it forms exactly a string of 2^{20} octets (1,048,576 octets), truncating the last repetition, if necessary. This result is called *digest0* [Stallings, 1998]. In the second step, the *digest0* is hashed using either the MD5 or the SHA-1 algorithm to derive *digest1*. MD5 algorithm yields a 16-octet *digest1,* and SHA-1 results in a 20-octet *digest1*. A second string is formed by concatenating the authoritative SNMP engine ID and *digest1*. This string is fed into the respective hashing algorithm to derive *digest2.* The derived *digest2* is the user's authentication key, *authKey*, which is input to the authentication modules shown in Figures 7.13 and 7.14. You are referred to RFC [2104] and Stallings [1998] for details on MD5 and SHA-1 algorithms.

The choice between the 16-octet MD5-based *authKey* and the 20-octet SHA-1-based *authKey* is based on the implementation. In the 20-octet key, it is harder to break the code than in the 16-octet key. However, the processing is faster with the 16-octet key. Further, the same 16-octet key derived from the same password could be used for the privacy key, although it is recommended that the same key not be used for both.

HMAC Procedure. The 96-bit long code MAC is derived using the HMAC procedure described in RFC 2104 and RFC 2274. First, two functions K1 and K2 are derived using *authKey* obtained above, and two fixed but different strings, *ipad* and *opad,* as defined in the following manner. A 64-byte *extendedAuthKey* is derived by supplementing *authKey* with zeros.

ipad = the hexadecimal byte 0x36 (00110110) repeated 64 times

opad = the hexadecimal byte 0x5c (01011100) repeated 64 times

K1 = *extendedAuthKey* XOR *ipad*

K2 = *extendedAuthKey* XOR *opad*

HMAC is computed by performing the following nested hashing functions on K1, K2, and *wholeMsg*, which is the unauthenticated whole message shown in Figure 7.13:

H (K2, H (K1, *wholeMsg*))

The first 12 octets of this final digest are the MAC. These are the authentication parameters, *msgAuthenticationParameters,* which are shown in Figure 7.12 and are included as part of the authenticated whole message, *authenticatedWholeMsg* shown in Figure 7.13.

Key Management. A user (NMS) has only one password and hence one *secret* key *digest1* mentioned in the authentication key discussion earlier. However, it communicates with all the authoritative SNMP engines (all the agents in the network). The shared information is again a secret between the two communicating engines. The concept of a localized key is introduced to accomplish this instead of storing a separate password for each pair of communicating engines. A hash function, which is the same hashing function that is used to generate the secret key, is employed to generate the localized key.

Localized key = H (*secret, authoritativeSnmpEngineID, secret)*

where *secret* is the secret key (*digest1*) and the *authoritativeSnmpEngineID* is the SNMP engine ID of the authoritative SNMP engine with which the local user is communicating. This localized key, different for each authoritative engine, is stored in each authoritative engine with which the user communicates. Notice that the localized key is the same as *authKey*.

SNMPv3 permits the operation of changes and modification in a key, but not the creation of keys to ensure that the secret key does not become stale.

Discovery. One important function of an NMS as a user is the discovery of agents in the network. This is accomplished by generating a Request message with a security level of no-authentication and no-privacy, a user name of "initial," an authoritative SNMP engine ID of zero length, and a varBind list that is empty. The authoritative engines respond with Response messages containing the engine ID and the security parameters filled in. Additional information is then obtained via pair-wise communication messages.

7.7.2 Encryption Protocol

The encryption generates non-readable *ciphertext* from a readable text, *plaintext.* The SNMPv3 recommendation for data confidentiality is to use the CBC-DES Symmetric Encryption Protocol. The USM specifications require the *scopedPDU* portion of the message be encrypted. A secret value in combination with a timeliness value is used to create the encryption/decryption key and initialization vector (IV). Again, the secret value is user-based, and hence is associated typically with an NMS. The 16-octet privacy key, *privKey,* is generated from the password as described in the generation of authentication code using MD-5 hashing algorithm.

The first eight octets of the 16-octet privacy key are used to create the DES key. The DES key is only 56 bits long and hence the least significant bit of each octet in the privacy key is discarded. The 16-octet IV is made up of two parts, an 8-octet pre-IV concatenated with an 8-octet *salt.* The pre-IV is the last eight octets of the privacy key. The *salt* is added to ensure that two identical instances of ciphertext are not generated from two different plaintexts using the same key. The *salt* is generated by an SNMP engine by concatenating a 4-octet *snmpEngineBoots* with a locally generated integer. The *salt* is the privacy parameter shown in Figures 7.12, 7.13, and 7.14.

The encryption process first divides the plaintext of *scopedPDU* into 64-bit blocks. The plaintext of each block is XOR-ed with the ciphertext of the previous block, and the result is encrypted to produce a ciphertext for the current block. For the first block, the IV is used instead of the ciphertext of the previous block.

7.8 ACCESS CONTROL

We have covered security considerations in network management with regard to data integrity, message authentication, data confidentiality, and the timeliness of message in the previous two sections. We will now address access control, which deals with who can access network management components and what they can access. In SNMPv1 and SNMPv2, this subject has been covered using the community-based access policy. In SNMPv3, access control has been made more secure and more flexible. It is called VACM.

VACM defines a set of services that an application in an agent can use to validate command requests and notification receivers. It validates command requests as to the sending sources and their access privilege. It assumes that authentication of the source has been done by the authentication module. In order to perform the services, a local database containing access rights and policies has been created in the SNMP entity, called Local Configuration Datastore (LCD). This is typically in an agent or in a manager functioning in an agent role when it communicates with another manager.

The LCD needs to be configured remotely and hence security considerations need to be introduced. A MIB module for VACM has been introduced toward achieving this.

7.8.1 Elements of the Model

Five elements comprise VACM: (1) groups, (2) security level, (3) contexts, (4) MIB views and view families, and (5) access policy. We will define each of them now.

Groups. A group, identified as *groupName*, is a set of zero or more SM (*vacmSecurityModel*)—security name (*vacmSecurityName*) pairs on whose behalf SNMP management objects can be accessed. A security name is a principal as defined in Section 7.3.2 and is independent of the SM used. All elements belonging to a group have identical access rights. Equivalent of a group in SNMPv1 is the community name. Thus, all NMSs (security names) in SNMPv1 (SM) with a community name public (group) would have equal access privilege to an agent.

Security Level. Security level (*vacmAccessSecurityLevel*) is the level of security of the user, namely no authentication–no privacy, authentication–no privacy, and authentication–privacy. This is set by the message flag shown in Figure 7.12. A member using a specific SM and with a given security name in a group could have different access rights by using different security levels.

Contexts. As mentioned in Section 7.3, an SNMP context is a collection of management information accessible by an SNMP entity. An SNMP entity has access to potentially more than one context. Each SNMP engine has a context table that lists the locally available contexts by *contextName*.

MIB Views and View Families. As in SNMPv1 and SNMPv2, access rights to contexts are controlled by a MIB view (see Figure 5.2). A MIB view is defined for each group and it details the set of managed object types (and optionally, the specific instances of object types). Following the approach of the

tree-like naming structure for MIB, the MIB view is defined as a combination of a set of *view subtrees*, where each view subtree is a subtree within the managed object naming tree. A simple MIB view could be all nodes defined under an OBJECT IDENTIFIER, for example, *system*. A view subtree is identified by the OBJECT IDENTIFIER value, which is the longest OBJECT IDENTIFIER prefix common to all (potential) MIB object instances in that subtree. For the system example, it is {1.3.6.1.2.1.1}.

An example of a complex MIB view could be all information relevant to a particular network interface. This can be represented by the union of multiple view subtrees, such as a set of *system* and *interfaces* to view all managed objects under the System and Interfaces groups.

A more complex view is a situation where all the columnar objects in a conceptual row of a table appear in separate subtrees, one per column, each with a similar format. Because the formats are similar, the required set of subtrees can be aggregated into one structure, called a *family of view subtrees*. A family of view subtrees is a pairing of an OBJECT IDENTIFIER value (called the family name) together with a bit string value (called the family mask). The family mask indicates which subidentifiers of the associated family name are significant to the family's definition. A family of view subtrees can either be included or excluded from the MIB view.

Access Policy: The access policy determines the access rights to objects as *read-view, write-view,* and *notify-view*. For a given *groupName*, *contextName*, *securityModel*, and *securityLevel*, that group's access rights are defined by either the combination of the three views, or *not-accessible*. The read-view is used for get-request, get-next-request, and get-bulk-request operations. The write-view is used with the set-request operation. The notify-view represents the set of object instances authorized for the group when sending objects in a notification.

7.8.2 VACM Process

The VACM process is presented as a flowchart in Figure 7.16. We will explain the process in terms of an SNMP agent with an SNMP engine having responsibility for many contexts. The tables shown in the figure are addressed in the next section on VACM MIB. As RFC 2275 describes, the VACM process answers the six questions related to access of management information. They are:

1. Who are you (group comprising security model and security name)?
2. Where do you want to go (context to be accessed)?
3. How secured are you to access the information (security model and security level)?
4. Why do you want to access the information (to read, write, or send notification)?
5. What object (object type) do you want to access?
6. Which object (object instance) do you want to access?

The first question is answered by the introduction of the group concept. The group that the requester belongs to is determined by the VACM from the SM and the security name. It uses the security-to-group table for validating the principal and deriving the group name.

The second question is answered by checking whether the context that needs to be accessed is within the responsibility of the agent. If the first two questions are answered in the affirmative, then the results of those, namely *group name* and *context name,* along with the *security model* and the *security level* (answer to how), are fed into the "access allowed?" process. It is assumed by VACM that the SM and the security level (the third question) have already been validated by the security module. Given these four

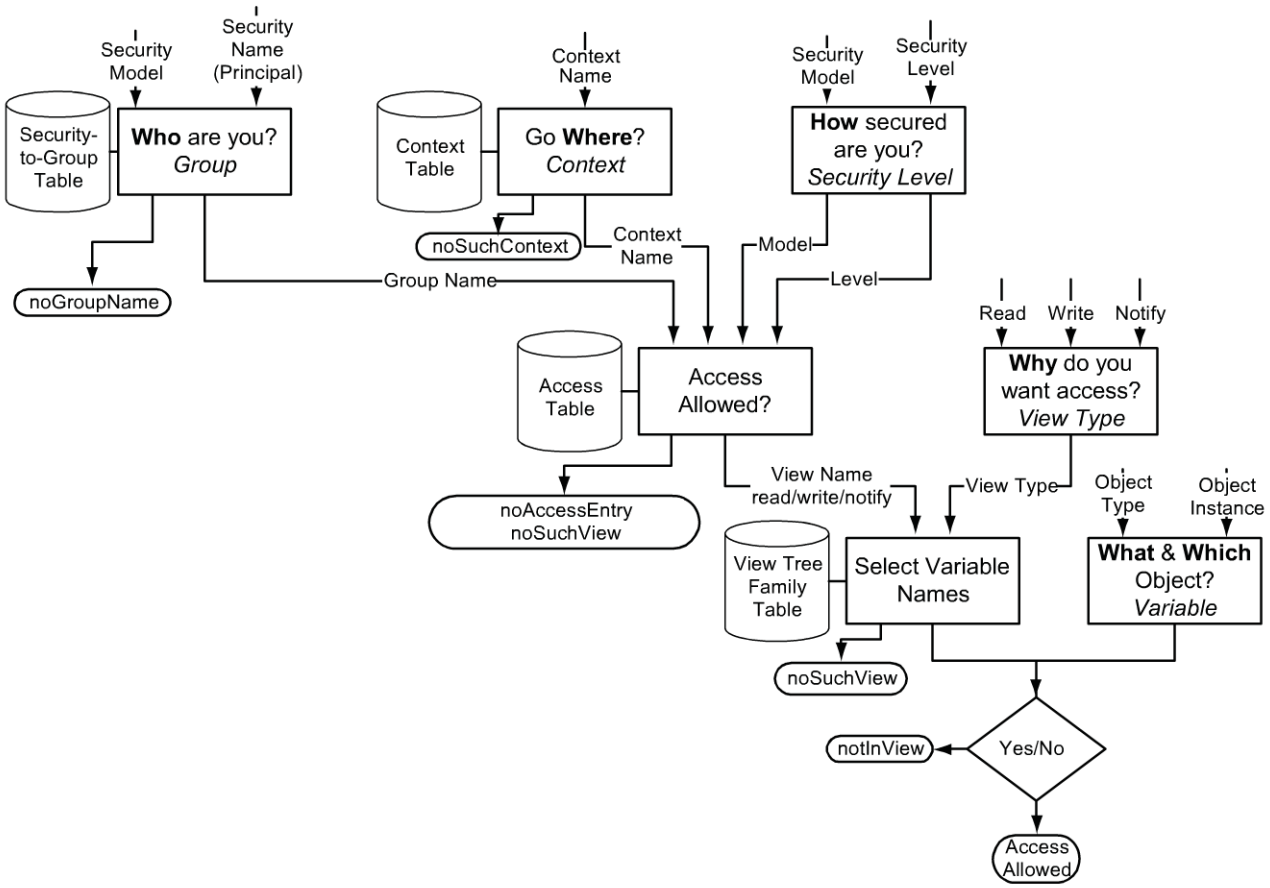

Figure 7.16 VACM Process

inputs as indices, the access table provides the views permitted, *view name*. It comprises one or more of the views, read-view, write-view, and notify-view.

The answer to the fourth question regarding why access is needed is used by the "select variable names" process to select the family of view subtrees eligible to be accessed. The view tree family table is applicable for this selection. A match is made between the result of this process and the answers to the last two questions as to what (object type) and which (object instance), to make a decision on whether access is allowed or not.

Each process puts out an error message based on the validation as shown in Figure 7.16.

7.8.3 VACM MIB

The processes in VACM use the tables to perform the functions mentioned. A VACM MIB has been defined specifying the newly created objects. This is shown in Figure 7.17. The *snmpVacmMIB* is a node under *snmpModules* shown in Figure 7.7. The three tables defining the context, group, and access are nodes under *vacmMIBObjects*, which is a node under *snmpVacmMIB*.

The *vacmContextTable* is a list of *vacmContextNames*. The *vacmSecurityToGroupTable* has columnar objects, *vacmSecurityModel*, *vacmSecurityName*, as indices to retrieve *vacmGroupName*.

The VACM Access Table, shown in Figure 7.18, is used to determine the access permission and the *viewName*. It has *vacmGroupName* from the *vacmSecurityToGroupTable* as one of the indices.

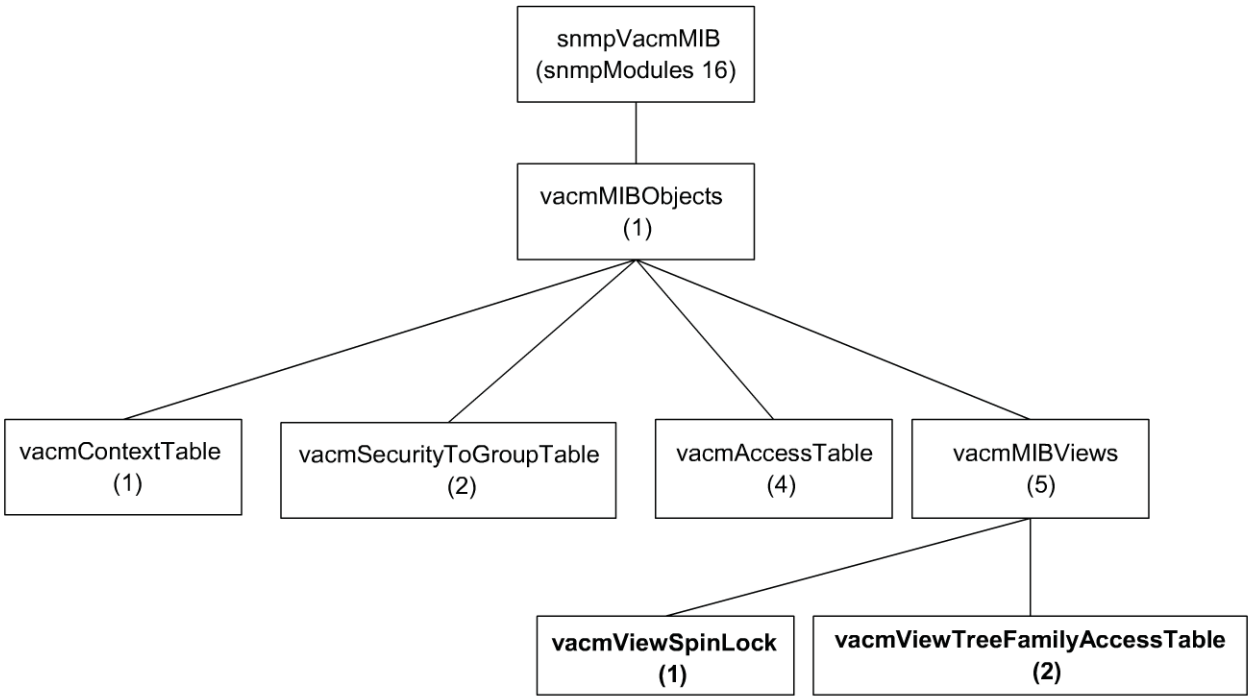

Figure 7.17 VACM MIB

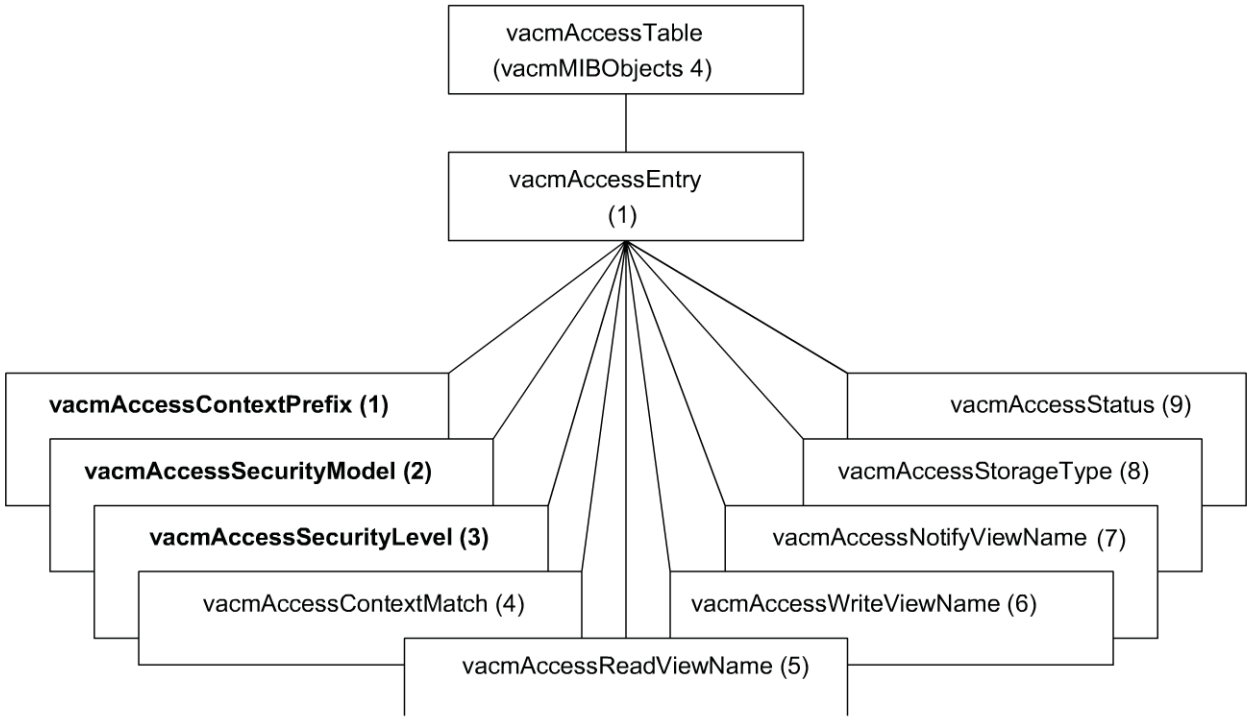

Figure 7.18 VACM Access Table

The other three indices from this table are *vacmAccessContextPrefix*, *vacmAccessSecurityModel*, and *vacmAccessSecurityLevel*. The *viewName* representing the three views, *vacmAccessReadViewName*, *vacmAccessWriteViewName*, and *vacmAccessNotifyViewName*, are retrieved from the table. The

vacmAccessStorageType and *vacmAccessStatus* are the administrative information objects relating to the storage volatility and the row status.

The *vacmMIBViews*, subnode (5), under *vacmMIBObjects*, shown in Figure 7.19, has the subordinate nodes *vacmViewSpinLock* and *vacmViewTreeFamilyAccessTable*. The *vacmViewSpinLock* is an advisory lock that is used by SNMP command generator applications to coordinate their use of the set operation in creating or modifying views in agents. It is an optional implementation object.

The *vacmViewTreeFamilyTable* describes families of subtrees that are available within MIB views in the local SNMP agent for each context. Each row in this table describes a subtree for a *viewName* and an OBJECT IDENTIFIER. For example, if the "access allowed?" process in Figure 7.16 yields three values for *viewName*, that would result in three conceptual rows in this table. The *vacmViewTreeFamilyViewName* representing the *viewName* is one of the columnar objects and an index in the table. Two indices define a conceptual row in this table. The second is *vacmViewTreeFamilySubtree*. It is a node representing the top of the tree. For example, if the OBJECT IDENTIFIER were 1.3.6.1.2.1.1, it would represent the *system* subtree. The OBJECT IDENTIFIER for the local agent is determined by the highest OBJECT IDENTIFIER that would address all object instances in the local view.

In some situations, we may want to view different subsets of a subtree. In such cases, we can form a family of view subtrees by using a combination of two parameters. The first is the selection of the view, which is done by a family mask defined by *vacmViewTreeFamilyMask*; and the second parameter is the family type defined by *vacmViewTreeFamilyType*, shown in Figure 7.19. The family mask is a bit string that is used with the *vacmViewTreeFamilySubtree*. Using this feature, specific objects in a subtree are selected if the corresponding object identifier matches. If the corresponding bit value is 0 in the family mask, it is considered a wild card and any value of the object identifier would be

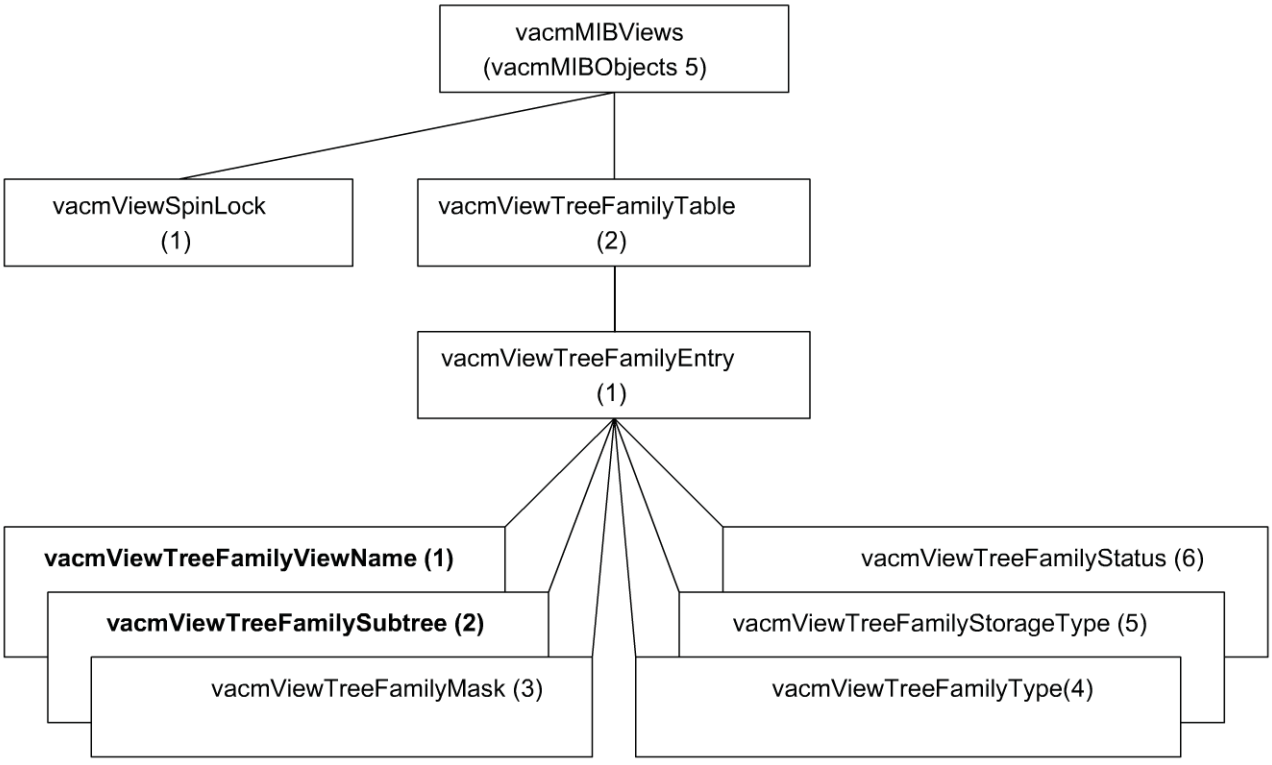

Figure 7.19 VACM MIB Views

selected. After the selection is made, if the family type value is included (1), the view is included. If it is excluded (2), the view is excluded. There is more flexibility in the views by introducing a columnar object *vacmViewTreeFamilyType* that indicates whether a particular subtree in the family of subtrees derived from *vacmViewTreeFamilySubtree* and *vacmViewTreeFamilyMask* is to be included or excluded in a context's MIB view.

As an example of the System group to be included, values for various parameters of the family entry in Figure 7.19 are:

Family view name = "system"
Family subtree = 1.3.6.1.2.1.1
Family mask = ""
Family type = 1

The zero length string, "", for mask value designates all 1s by convention.

We could extend the view by adding a second row to the table. For example, we could add an SNMP group by adding another row to the table with the family subtree 1.3.6.1.2.1.11.

Suppose we want to add a columnar object to the table. We would add the columnar object with the index added as another row. We could also add all the columnar objects of a conceptual row in a table. A useful convention for doing this is to use the definition of columnar object 0, which designates all columnar objects in a table. For example, {1.3.6.1.2.1.2.2.1.0.5} identifies all columnar objects associated with the 5th interface (corresponding to *ifIndex* value of 5) in the *ifTable*.

If more than one family name is present with the same number of subidentifiers, the lexicographic convention is followed for the predominance among them. This helps in the following way. Suppose we wanted to choose all columnar objects in the above *ifTable* example, except the *ifMtu*, which is the 4th columnar object. We would then choose {1.3.6.1.2.1.2.2.1.4.5} and the Type = 2 to exclude it. Since this is lexicographically higher than {1.3.6.1.2.1.2.2.1.0.5}, this will take precedence. Thus, the combination of the two will select all 5th row objects except *ifMtu*.

Summary

We have reviewed the latest version of SNMP, SNMPv3, in this chapter. The two major features are the specifications for a formalized SNMP architecture that addresses the three SNMP frameworks for the three versions. Two new members, dispatch and message processing modules, are defined. This would enable a network management system to handle messages from and to agents that belong to all three current versions. It would also accommodate future versions, if needed.

The second major feature is the inclusion of security. A security subsystem is defined, which addresses data integrity, data origin authentication, data confidentiality, message timeliness, and limited message replay protection. The authentication module in the security subsystem addresses the first two issues, the privacy module protects data confidentiality, and the timeliness module deals with message timeliness and limited replay protection. The security subsystem is the User-based Security Model (USM). It is derived from the traditional concept of user ID and password.

The access policies of SNMPv1 and SNMPv2 have been extended and made more flexible by the VACM. An SNMP agent handling multiple objects (contexts) can be configured to present a set of MIB views and a family of subtrees in its MIB views. These views can be matched with seven input parameters to determine access permission to the principal. They are the SM (version of SNMP), the security name (principal), the security level (dependent on the authentication and privacy parameters), the context name, the type of access needed, the object type, and the object instance.

Exercises

1. The first four octets of an SNMP engine ID in a system are set to the binary equivalent of the system's SNMP management private enterprise number as assigned by the IANA. Write the first four octets of the SNMP engine ID in hexadecimal notation for the four enterprises, (cisco, hp, 3com, and cabletron,) shown in Figure 4.14 for the following two versions:
   (a) SNMPv1
   (b) SNMPv3
2. Write the full SNMP engine ID for:
   (a) SNMPv1 for a 3Com hub with the IPv4 address 128.64.46.2 in the 6th to 9th octets followed by 0s in the rest.
   (b) SNMPv3 for the Cisco router interface with IPv6 address ::128.64.32.16.
3. Describe the SNMPv3 *scopedPDU* that the SNMP agent (router) responds to NMS with the data shown in Figure 4.2(c).
4. Figure 7.20 shows a generalized time-sequenced operation for get-request message going from a manager to an agent. Complete the primitives in Figure 7.20 explicitly identifying the application modules used.

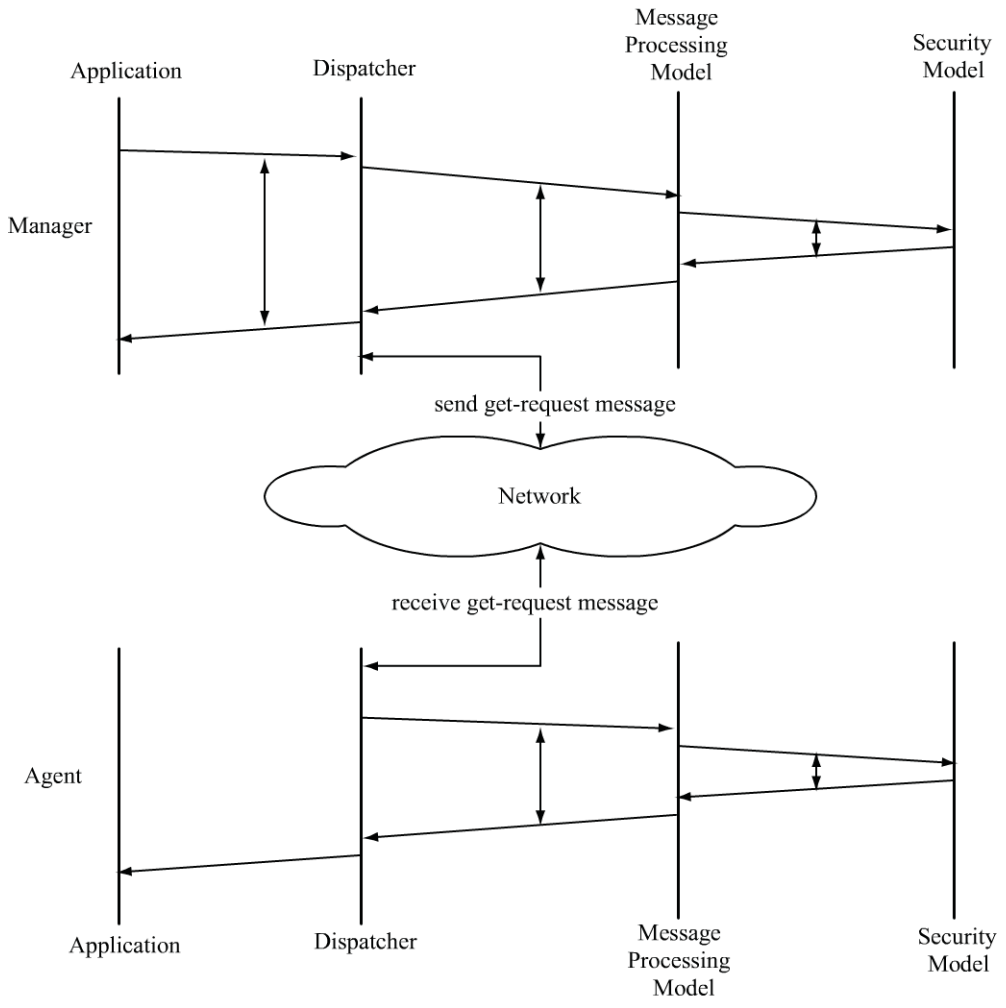

Figure 7.20 Exercise 4

5. Draw the time-sequence operation similar to that in Figure 7.20 detailing the elements of procedure for get-response message from the agent to the manager.
6. Detail the IN and OUT parameters of the *sendPdu* and *prepareOugoingMsg* primitives shown in Figure 7.4(b) by referring to RFC 2271.
7. Identify the authoritative and non-authoritative entities in Figure 7.20.
8. Define the configuration parameters for a notification generator to send traps to two network management systems *noc1* and *noc2* by filling in the objects in the *snmpTargetAddressTable*, *snmpTargetTable*, and *snmpNotifyTable*. Specifications for the two targets are given below. You may use the Appendix of RFC 2273 as a guide to answer this exercise.

	noc1	noc2
messageProcessingModel	SNMPv3	SNMPv3
securityModel	3 (USM)	3 (USM)
secuirtyName	“noc1”	“noc2”
snmpTargetParamsName	“NOAuthNoPriv-noc1”	“NOAuthNoPriv-noc1”
securityLevel	noAuthNoProv(1)	authPriv(3)
transportDomain	snmpUDPDomain	snmpUDPDomain
transportAddress	128.64.32.16:162	128.64.32.8:162
tagList	“group1”	“group2”

9. Access RFC 2274 and list and define the primitives provided by the authentication module at the sending and receiving security subsystems. Describe the services provided by the primitives.
10. Access RFC 2274 and list and define primitives provided by the privacy module at the sending and receiving security subsystems. Describe the services provided by the primitives.
11. Specify the family name, the family subtree, the family mask, and the family type in *vacmViewTreeFamilyTable* for an agent to present a view of:
    (a) the complete IP group
    (b) IP address table (*ipAddrTable*)
    (c) the row in the IP address table corresponding to the IP address 172.46.62.1
12. Write the *vacmViewTreeFamilyTable* for the three rows that present the system group in the IP address table for the row with IP address 172.46.62.1 without the *ipAdEntReasmMaxSize*.