IEEE COMMUNICATIONS SURVEYS

SNMPv3: A Security Enhancement for SNMP

William Stallings

Abstract

Simple Network Management Protocol (SNMP) is the most widely-used network management protocol on TCP/IP-based networks. The functionality of SNMP was enhanced with the publication of SNMPv2. However, both these versions of SNMP lack security features, notably authentication and privacy, that are required to fully exploit SNMP. A recent set of RFCs, known collectively as SNMPv3, correct this deficiency. This article outlines the overall network management framework defined in SNMPv3, and then looks at the principal security facilities defined in SNMPv3: authentication, privacy, and access control.

Since its first publication in 1988, the Simple Network Management Protocol (SNMP) has become the most widely-used network-management tool for TCP/IP-based networks. SNMP defines a protocol for the exchange of management information, but does much more than that. It also defines a format for representing management information and a framework for organizing distributing systems into managing systems and managed agents. In addition, a number of specific data base structures, called management information bases (MIBs), have been defined as part of the SNMP suite; these MIBs specify managed objects for the most common network management subjects, including bridges, routers, and LANs.

The rapid growth in the popularity of SNMP in the late 1980s and early 1990s led to an awareness of its deficiencies; these fall into the broad categories of functional deficiencies, such as the inability to easily specify the transfer of bulk data, and security deficiencies, such as the lack of authentication and privacy mechanisms. Many of the functional deficiencies were addressed in a new version of SNMP, known as SNMPv2, first published as a set of RFCs in 1993. The 1993 edition of SNMPv2 also included a security facility, but this was not widely accepted because of a lack of consensus and because of perceived deficiencies in the definition. Accordingly, a revised edition of SNMPv2 was issued in 1996, with the functional enhancements intact but without a security facility. This version used the simple and unsecure password-based authentication feature, known as the community feature, provided in SNMPv1, and is referred to as SNMPv2c. To remedy the lack of security, a number of independent groups began work on a security enhancement to SNMPv2. Two competing approaches emerged as front-runners: SNMPv2u and SNMPv2*. Ultimately, these two approaches served as input to a new IETF SNMPv3 working group, which was chartered in March of 1997. By January of 1998, this group had produced a set of Proposed Internet standards published as RFCs 2271–2275 (Table 1). This document set defines a framework for incorporating security features into an overall capability that includes either SNMPv1 or SNMPv2 functionality. In addition, the documents defines a specific set of capabilities for network security and access control.

It is important to realize that SNMPv3 is not a stand-alone replacement for SNMPv1 and/or SNMPv2. SNMPv3 defines a security capability to be used in conjunction with SNMPv2 (preferred) or SNMPv1. In addition, RFC 2271, which is one of the documents issued by the SNMPv3 working group, describes an architecture within which all current and future versions of SNMP fit. RFC 2275 describes an access control facility, which is intended to operate independently of the core SNMPv3 capability. Thus, only three of the five documents issued by the SNMPv3 working group deal with SNMPv3 security. In this article, we take a broader view and provide a survey of the capabilities defined in RFCs 2271 through 2275.

Figure 1 describes the relationship among the different versions of SNMP by means of the formats involved. Information is exchanged between a management station and an agent in the form of an SNMP message. Security-related processing occurs at the message level; for example, SNMPv3 specifies a User Security Model (USM) that makes use of fields in the message header. The payload of an SNMP message is either an SNMPv1 or an SNMPv2 protocol data unit (PDU). A PDU indicates a type of management action (e.g., get or set a managed object) and a list of variable names related to that action.

A brief clarification of the term SNMPv3 is perhaps in order. RFCs 2271 through 2275, produced by the SNMPv3 working group describe an overall architecture plus specific message structures and security features, but do not define a new SNMP PDU format. Thus, the existing SNMPv1 or SNMPv2 PDU format must be used within the new architecture. An implementation referred to as SNMPv3 consists of the security and architectural features defined in RFCs 2271 through 2275 plus the PDU format and functionality defined in the SNMPv2 documents. This is expressed in the SNMPv3 introduction document, Table 15.1, as follows: "SNMPv3 is SNMPv2 plus security and administration."

The remainder of this article is organized as follows. The next section provides a brief introduction to the basic SNMP concepts. This is followed by a discussion of the SNMP architecture defined in RFC 2271. Next, the privacy and authentication facilities provided by the SNMPv3 User Security Model (USM) are described. The next section discusses access control and the view-based access control model (VACM). An appendix provides a brief tutorial on cryptographic algorithms, including encryption and message authentication.

Number	Title	Date
RFC 2271	An Architecture for Describing SNMP Management Frameworks	January 1998
RFC 2272	Message Processing and Dispatching for SNMP	January 1998
RFC 2273	SNMPv3 Applications	January 1998
RFC 2274	User-Based Security Model for SNMPv3	January 1998
RFC 2275	View-Based Access Control Model (VACM) for SNMP	January 1998
Internet Draft	Introduction to Version 3 of the Internet Network Management Framework	August 1998

■ **Table 1.** *SNMPv3 documents.*

Basic SNMP Concepts

The basic idea of any network management system is that there are two types of systems in any networked configuration: agents and managers. Any node in the network that is to be managed, including PCs, workstations, servers, bridges, routers, and so on, includes an agent module. The agent is responsible for

- Collecting and maintaining information about its local environment
- Providing that information to a manager, either in response to a request or in an unsolicited fashion when something noteworthy happens
- Responding to manager commands to alter the local configuration or operating parameters.

A configuration will also include one or more management stations, or managers. The manager station generally provides a user interface so that a human network manager can control and observe the network management process. This interface allows the user to issue commands (e.g., deactivate a link, collect statistics on performance, etc.) and provides logic for summarizing and formatting information collected by the system.

The heart of the network management system is a set of applications that meet the needs for network management. At a minimum, a system will include basic applications for performance monitoring, configuration control, and accounting. More sophisticated systems will include more elaborate applications in those categories, plus facilities for fault isolation and correction, and for managing the security features of the network.

All of the network management applications generally share a common network management protocol. This protocol provides the fundamental functions for retrieving management information from agents and for issuing commands to agents. This protocol, in turn, makes use of a communications facility, such as TCP/IP or OSI.

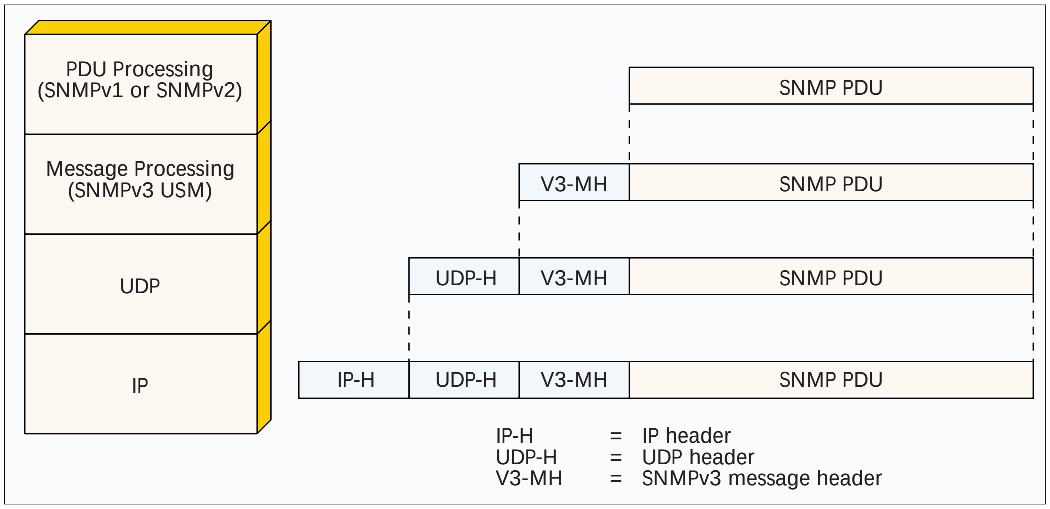
Finally, each agent maintains a management information base (MIB) that contains current and historical information about its local configuration and traffic. The management station will maintain a global MIB with summary information from all the agents.

The SNMPv1 and SNMPv2 specifications consists of a set of documents that define a network management protocol, a general structure for management information bases (MIBs), and a number of specific MIB data structures for specific management purposes. The specification includes minimal network management applications and no user presentation facility. Thus, SNMP is not a full-blown network management standard. Accordingly, vendors have provided their own proprietary network management applications to run on top of SNMP.

The operative word in SNMP is "simple." SNMP is designed to be easy to implement and to consume minimal processor and network resources. It is therefore a tool for building a bare-bones network management facility. In essence the protocol provides four functions:

- Get: Used by a manager to retrieve an item from an agent's MIB.
- Set: Used by a manager to set a value in an agent's MIB.
- Trap: Used by an agent to send an alert to a manager.
- Inform: Used by a manager to send an alert to another manager.

This is about as simple as you can get. What gives SNMP its power is the extensive set of standardized MIB structures that has been defined. The MIB at an agent dictates what information that agent will collect and store. For example, there are a number of variables in the basic MIB that relate to the operation of the underlying TCP and IP protocols, including number of packets sent and received, packets in error, and so on. Since all agent maintain the same set of data variables, applications can be written at the management station to exploit this information.



■ **Figure 1.** *SNMP protocol architecture.*

For a more detailed discussion of SNMPv1 and SNMPv2, see reference [1].

SNMP ARCHITECTURE

The SNMP architecture, as envisioned by RFC 2271, consists of a distributed, interacting collection of SNMP entities. Each entity implements a portion of the SNMP capability and may act as an agent node, a manager node, or a combination of the two. Each SNMP entity consists of a collection of modules that interact with each other to provide services. These interactions can be modeled as a set of abstract primitives and parameters.

The RFC 2271 architecture reflects a key design requirement for SNMPv3: Design a modular architecture that will

- Allow implementation over a wide range of operational environments, some of which need minimal, inexpensive functionality and some of which may support additional features for managing large networks
- Make it possible to move portions of the architecture forward in the standards track even if consensus has not been reached on all pieces
- Accommodate alternative security models

SNMP ENTITY

Each SNMP entity includes a single SNMP engine. An SNMP engine implements functions for sending and receiving messages, authenticating and encrypting/decrypting messages, and controlling access to managed objects. These functions are provided as services to one or more applications that are configured with the SNMP engine to form an SNMP entity.

Both the SNMP engine and the applications it supports are defined as a collection of discrete modules. This architecture provides several advantages. First, as we shall see, the role of an SNMP entity is determined by which modules are implemented in that entity. For example, a certain set of modules is required for an SNMP agent, while a different (though overlapping) set of modules is required for an SNMP manager. Second, the modular structure of the specification lends itself to defining different versions of each module. This in turn makes it possible to

- Define alternative or enhanced capabilities for certain aspects of SNMP without needing to go to a new version of the entire standard (e.g., SNMPv4)
- Clearly specify coexistence and transition strategies.

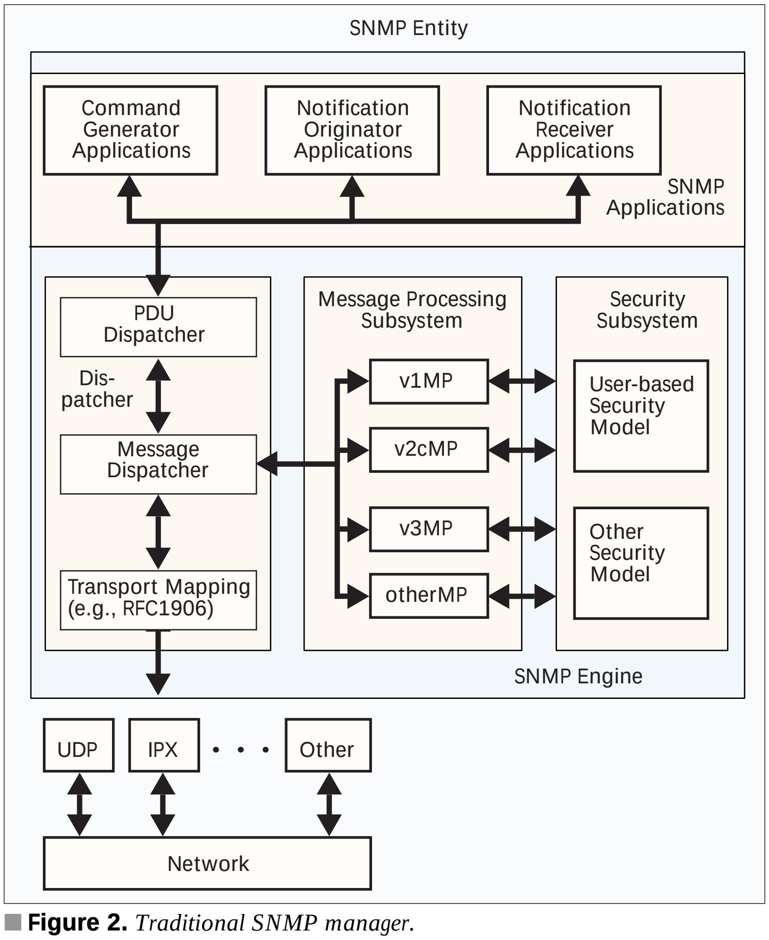
To get a better understanding of the role of each module and its relationship to other modules, it is best to look at their use in traditional SNMP managers and agents. The term traditional, equivalent to pure, is used to emphasize the fact that a given implementation need not be a pure manager or agent but may have modules that allow the entity to perform both management and agent tasks.

TRADITIONAL SNMP MANAGER — Figure 2, based on a figure in RFC 2271, is a block diagram of a traditional SNMP manager. A traditional SNMP manager interacts with SNMP agents by issuing commands (get, set) and by receiving trap message; the manager may also interact with other managers by issuing Inform Request PDUs, which provide alerts, and by receiving Inform Response PDUs, which acknowledge Inform Requests. In SNMPv3 terminology, a traditional SNMP manager includes three categories of applications. The **Command Generator Applications** monitor and manipulate management data at remote agents; they make use of SNMPv1 and/or SNMPv2 PDUs, including Get, GetNext, GetBulk, and Set. A **Notification Originator Application** initiates asynchronous messages; in the case of a traditional manager, the InformRequest PDU is used for this application. A **Notification Receiver Application** processes incoming asynchronous messages; these include InformRequest, SNMPv2-Trap, and SNMPv1 Trap PDUs. In the case of an incoming InformRequest PDU, the Notification Receiver Application will respond with a Response PDU.

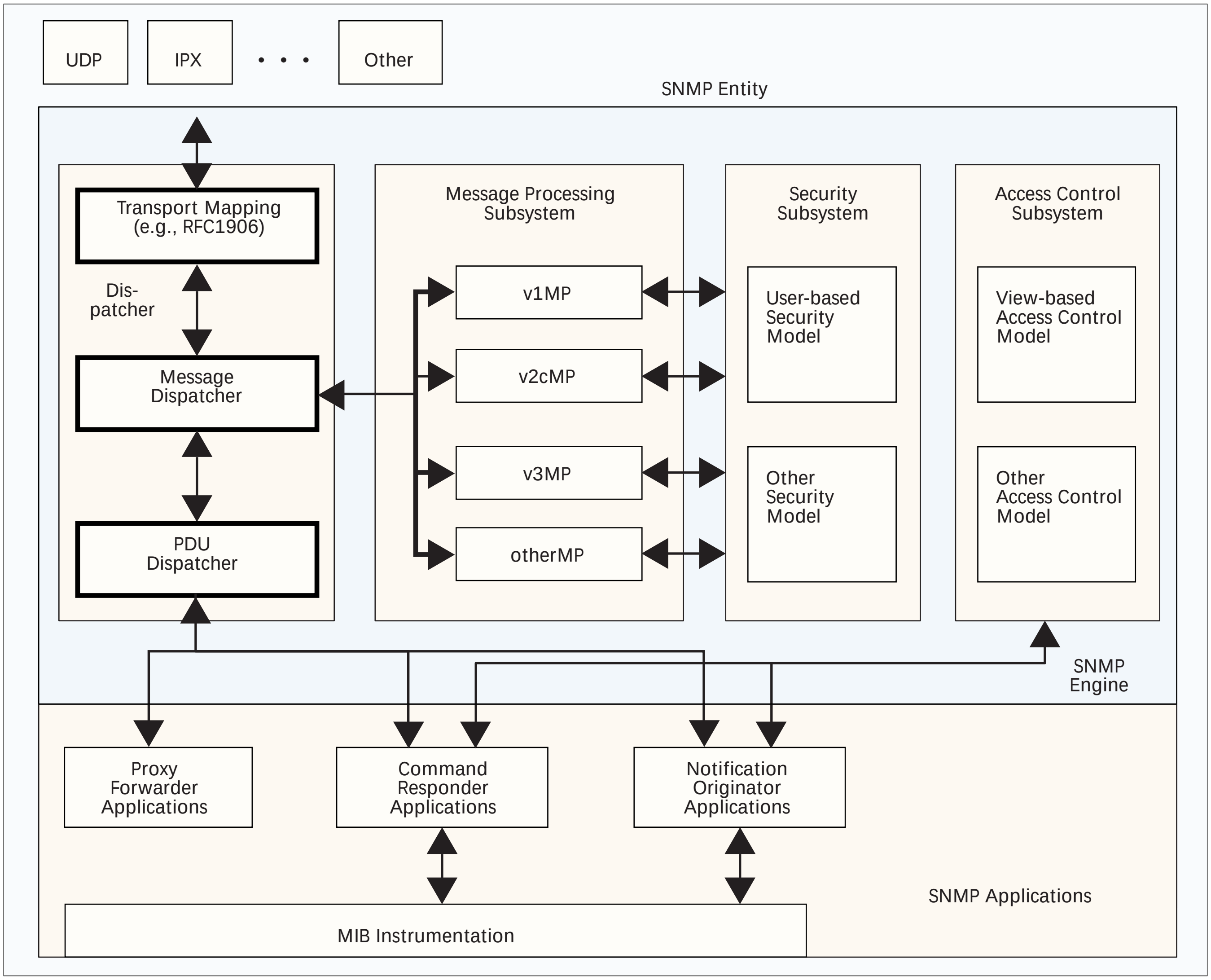
All of the applications just described make use of the services provided by the SNMP engine for this entity. The SNMP engine performs two overall functions:

- It accepts outgoing PDUs from SNMP applications, performs the necessary processing, including inserting authentication codes and encrypting, and then encapsulates the PDUs into messages for transmission.
- It accepts incoming SNMP messages from the transport layer, performs the necessary processing, including authentication and decryption, and then extracts the PDUs from the messages and passes these on to the appropriate SNMP application.

In a traditional manager, the SNMP engine contains a Dispatcher, a Message Processing Subsystem, and a Security Subsystem. The **Dispatcher** is a simple traffic manager. For outgoing PDUs, the Dispatcher accepts PDUs from applications and performs the following functions. For each PDU,


SNMP Entity
Command Generator Applications
Notification Originator Applications
Notification Receiver Applications
SNMP Applications
PDU Dispatcher
Dis-patcher
Message Dispatcher
Transport Mapping (e.g., RFC1906)
Message Processing Subsystem
v1MP
v2cMP
v3MP
otherMP
Security Subsystem
User-based Security Model
Other Security Model
SNMP Engine
UDP
IPX
• • •
Other
Network

■ **Figure 2.** *Traditional SNMP manager.*


UDP
IPX
• • •
Other
SNMP Entity
Transport Mapping (e.g., RFC1906)
Dis-patcher
Message Dispatcher
PDU Dispatcher
Message Processing Subsystem
v1MP
v2cMP
v3MP
otherMP
Security Subsystem
User-based Security Model
Other Security Model
Access Control Subsystem
View-based Access Control Model
Other Access Control Model
SNMP Engine
Proxy Forwarder Applications
Command Responder Applications
Notification Originator Applications
SNMP Applications
MIB Instrumentation

■ **Figure 3.** *Traditional SNMP agent.*

the Dispatcher determines the type of message processing required (i.e., for SNMPv1, SNMPv2c, or SNMPv3) and passes the PDU on to the appropriate message processing module in the Message Processing Subsystem. Subsequently, the Message Processing Subsystem returns a message containing that PDU and including the appropriate message headers. The Dispatcher then maps this message onto a transport layer for transmission.

For incoming messages, the Dispatcher accepts messages from the transport layer and performs the following functions. The Dispatcher routes each message to the appropriate message processing module. Subsequently, the Message Processing Subsystem returns the PDU contained in the message. The Dispatcher then passes this PDU to the appropriate application.

The **Message Processing Subsystem** accepts outgoing PDUs from the Dispatcher and prepares these for transmission by wrapping them in the appropriate message header and returning them to the Dispatcher. The Message Processing Subsystem also accepts incoming messages from the Dispatcher, processes each message header, and returns the enclosed PDU to the Dispatcher. An implementation of the Message Processing Subsystem may support a single message format corresponding to a single version of SNMP (SNMPv1, SNMPv2c, SNMPv3), or it may contain a number of modules, each supporting a different version of SNMP.

The **Security Subsystem** performs authentication and encryption functions. Each outgoing message is passed to the Security Subsystem from the Message Processing Subsystem. Depending on the services required, the Security Subsystem may encrypt the enclosed PDU and possibly some fields in the message header, and it may generate an authentication code and insert it into the message header. The processed message is then returned to the Message Processing Subsystem. Similarly, each incoming message is passed to the Security Subsystem from the Message Processing Subsystem. If required, the Security Subsystem checks the authentication code and performs decryption. It then returns the processed message to the Message Processing Subsystem. An implementation of the Security Subsystem may support one or more distinct security models. So far, the only defined security model is the User-Based Security Model (USM) for SNMPv3, specified in RFC 2274.

TRADITIONAL SNMP AGENT — Figure 3, based on a figure in RFC 2271, is a block diagram of a traditional SNMP agent. The traditional agent may contain three types of applications. Command Responder Applications provide access to management data. These applications respond to incoming requests by retrieving and/or setting managed objects and then issuing a Response PDU. A Notification Originator Application initiates asynchronous messages; in the case of a traditional agent, the SNMPv2-Trap or SNMPv1 Trap PDU is used for this application. A Proxy Forwarder Application forwards messages between entities.

The SNMP engine for a traditional agent has all of the components found in the SNMP engine for a traditional manager, plus an **Access Control Subsystem**. This subsystem provides authorization services to control access to MIBs for the reading and setting of management objects. These services are performed on the basis of the contents of PDUs. An implementation of the Security Subsystem may support one or more distinct access control models. So far, the only defined security model is the View-Based Access Control Model (VACM) for SNMPv3, specified in RFC 2275.

Note that security-related functions are organized into two separate subsystems: security and access control. This is an excellent example of good modular design, because the two subsystems perform quite distinct functions and therefore it makes sense to allow standardization of these two areas to proceed independently. The Security Subsystem is concerned with privacy and authentication, and operates on SNMP messages. The Access Control Subsystem is concerned with authorized access to management information, and operates on SNMP PDUs.

Terminology

Table 2 briefly defines some terms that are introduced in RFC 2271. Associated with each SNMP entity is a unique snmpEngineID. For purposes of access control, each SNMP entity is considered to manage a number of contexts of managed information, each of which has a contextName that is unique within that entity. To emphasize that there is a single manager of contexts within an entity, each entity has a unique contextEngineID associated with it; because there is a one-to-one correspondence between the context engine and the SNMP engine at this entity, the contextEngineID is identical in value to the snmpEngineID. Access control is governed by the specific context for which access is attempted and the identity of the user requesting access; this latter identity is expressed as a principal, which may be an individual or an application or a group of individuals or applications.

snmpEngineID	Unique and unambiguous identifier of an SNMP engine, as well as the SNMP entity that corresponds to that engine.
contextEngineID	Uniquely identifies an SNMP entity that may realize an instance of a context with a particular contextName.
contextName	Identifies a particular context within an SNMP engine. It is passed as a parameter to the Dispatcher and Access Control Subsystem.
scopedPDU	A block of data consisting of a contextEngineID, a contextName, and an SNMP PDU. It is passed as a parameter to/from the Security Subsystem.
snmpMessageProcessingModel	Unique identifier of a message processing model of the Message Processing Subsystem. Possible values include SNMPv1, SNMPv2c, and SNMPv3.
snmpSecurityModel	Unique identifier of a security model of the Security Subsystem. Possible values include SNMPv1, SNMPv2c, and USM.
snmpSecurityLevel	A level of security at which SNMP messages can be sent or with which operations are being processed, expressed in terms of whether or not authentication and/or privacy are provided. The alternative values are noAuthnoPriv, authNoPriv, and authPriv.
principal	The entity on whose behalf services are provided or processing takes place. A principal can be an individual acting in a particular role; a set of individuals, with each acting in a particular role; an application or set of applications; and combinations thereof.
securityName	A human-readable string representing a principal. It is passed as a parameter in all of the SNMP primitives (Dispatcher, Message Processing, Security, Access Control)

Table 2. *SNMPv3 terminology.*

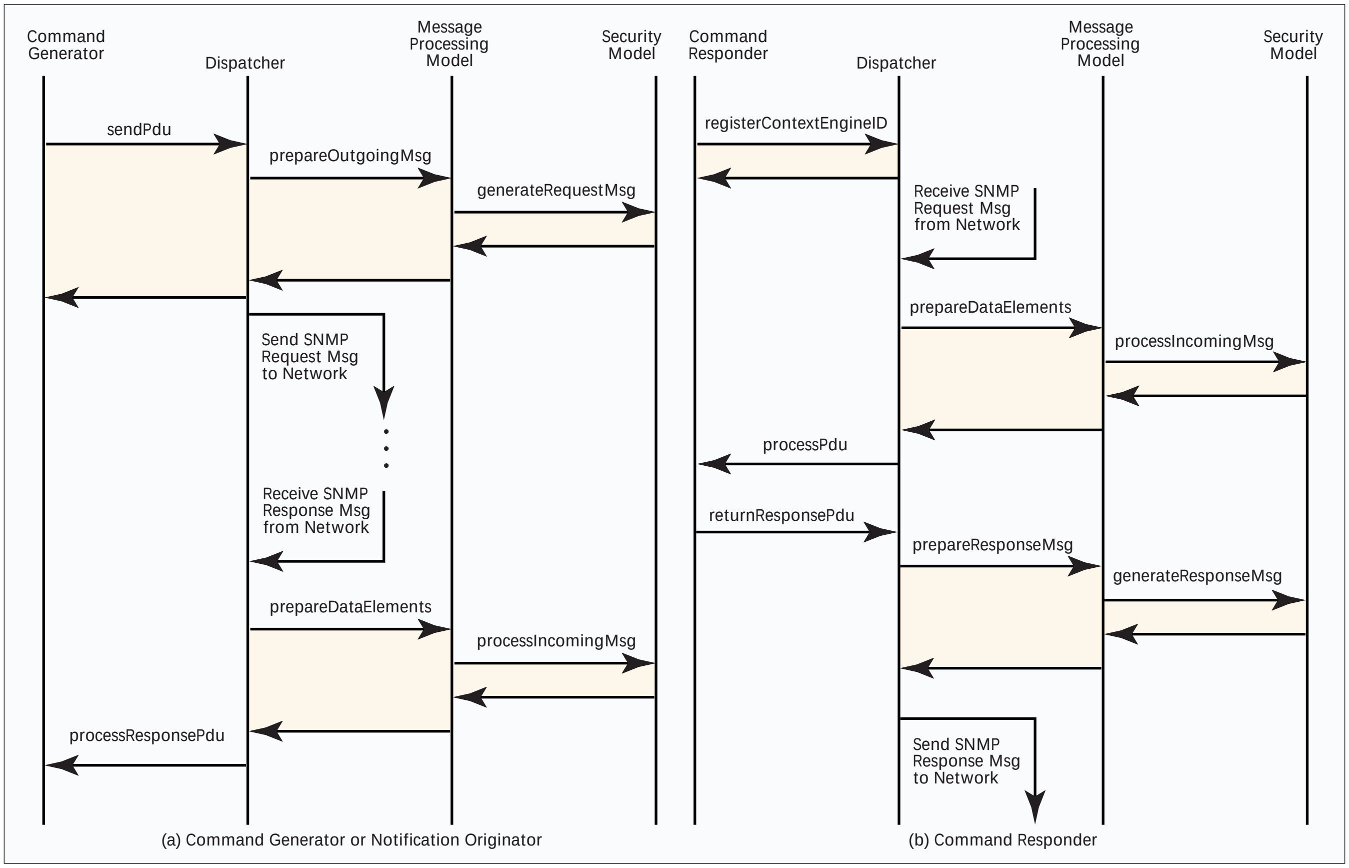
Other terms of importance relate to the processing of messages. The snmpMessageProcessingModel determines the message format and the SNMP version for message processing. The snmpSecurityModel determines which security model is to be used. The snmpSecurityLevel determines which security services are requested for this specific operation. The user may request just authentication, or authentication plus privacy (encryption), or neither.

SNMPv3 Applications

The services between modules in an SNMP entity are defined in the RFCs in terms of primitives and parameters. A primitive specifies the function to be performed, and the parameters are used to pass data and control information. We can think of these primitives and parameters as a formalized way of defining SNMP services. The actual form of a primitive is implementation dependent; an example is a procedure call. In the discussion that follows, it may be useful to refer to Fig. 4, based on a figure in RFC 2271, to see how all of these primitives fit together. Figure 4a shows the sequence of events in which a Command Generator or Notification Originator application requests that a PDU be sent, and subsequently how the matching response is returned to that application; these events occur at a manager. Figure 4b shows the corresponding events at an agent. The figure shows how an incoming message results in the dispatch of the enclosed PDU to an application, and how that application's response results in an outgoing message. Note that some of the arrows in the diagram are labeled with a primitive name, representing a call. Unlabeled arrows represents the return from a call, and the shading indicates the matching between call and return.

RFC 2273 defines, in general terms, the procedures followed by each type of application when generating PDUs for transmission or processing incoming PDUs. In all cases, the procedures are defined in terms of interaction with the Dispatcher by means of the Dispatcher primitives.

Command Generator Applications — A command generator application makes use of the sendPdu and processResponsePdu Dispatcher



■ **Figure 4.** *SNMPv3 flow.*

primitives. The sendPdu provides the Dispatcher with information about the intended destination, security parameters, and the actual PDU to be sent. The Dispatcher then invokes the Message Processing Model, which in turn invokes the Security Model, to prepare the message. The Dispatcher hands the prepared message over to the transport layer (e.g., UDP) for transmission. If message preparation fails, the return primitive value of the sendPdu, set by the Dispatcher, is an error indication. If message preparation succeeds, the Dispatcher assigns a sendPduHandle identifier to this PDU and returns that value to the command generator. The command generator stores the sendPduHandle so that it can match the subsequent response PDU to the original request.

The Dispatcher delivers each incoming response PDU to the correct command generator application, using the processResponsePdu primitive.

COMMAND RESPONDER APPLICATIONS — A command responder application makes use of four Dispatcher primitives (registerContextEngineID, unregisterContextEngineID, processPdu, returnResponsePdu), and one Access Control Subsystem primitive (isAccessAllowed).

The registerContextEngineID primitive enables a command responder application to associate itself with an SNMP engine for the purpose of processing certain PDU types for a context engine. Once a command responder has registered, all asynchronously received messages containing the registered combination of contextEngineID and pduType supported are sent to the command responder that registered to support that combination. A command responder can disassociate from an SNMP engine using the unregisterContextEngineID primitive.

The Dispatcher delivers each incoming request PDU to the correct command responder application, using the processPdu primitive. The command responder then performs the following steps:

1. The command responder examines the contents of the request PDU. The operation type must match one of the types previously registered by this application.
2. The command responder determines if access is allowed for performing the management operation requested in this PDU. For this purpose, the isAccessAllowed primitive is called. The securityModel parameter indicates which security model the Access Control Subsystem is to use in responding to this call. The Access Control Subsystem determines if the requesting principal (securityName) at this security level (securityLevel) has permission to request the management operation (viewType) on the management object (variableName) in this context (contextName).
3. If access is permitted, the command responder performs the management operation and prepares a response PDU. if access fails, the command responder prepares the appropriate response PDU to signal that failure.
4. The command responder calls the Dispatcher with a returnResponsePdu primitive to send the response PDU.

NOTIFICATION GENERATOR APPLICATIONS — A notification generator application follows the same general procedures used for a command generator application. If an Inform Request PDU is to be sent, both the sendPdu and processResponsePdu primitives are used, in the same fashion as for command generator applications. If a trap PDU is to be sent, only the sendPdu primitive is used.

NOTIFICATION RECEIVER APPLICATIONS — A notification receiver application follows a subset of the general procedures as for a command responder application. The notification receiver must first register to receive Inform and/or trap PDUs. Both types of PDUs are received by means of a processPdu primitive. For an Inform PDU, a returnResponsePdu primitive is used to respond.

PROXY FORWARDER APPLICATIONS — A proxy forwarder application makes use of Dispatcher primitives to forward SNMP messages. The proxy forwarder handles four basic types of messages:

- Messages containing PDU types from a command generator application. The proxy forwarder determines either the target SNMP engine or an SNMP engine that is closer, or downstream, to the target, and sends the appropriate request PDU.
- Messages containing PDU types from a notification originator application. The proxy forwarder determines which SNMP engines should receive the notification and sends the appropriate notification PDU or PDUs.
- Messages containing a Response PDU type. The proxy forwarder determines which previously forwarded request or notification, if any, is matched by this response, and sends the appropriate response PDU.
- Messages containing a report indication. Report PDUs are SNMPv3 engine-to-engine communications. The proxy forwarder determines which previously forwarded request or notification, if any, is matched by this report indication, and forwards the report indication back to the initiator of the request or notification.

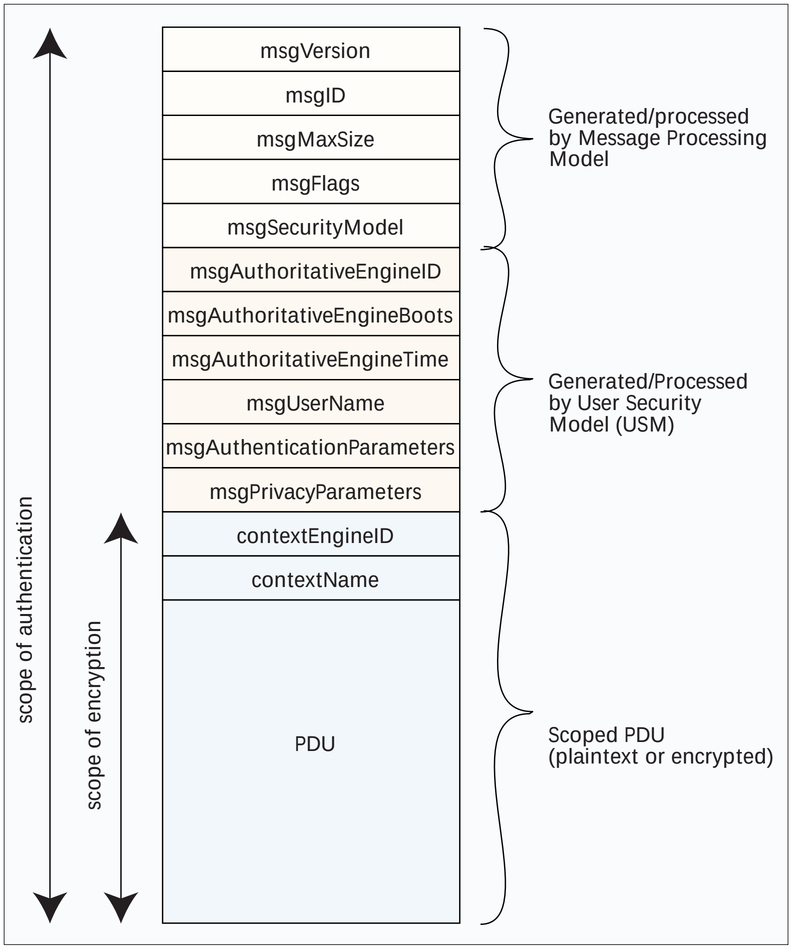

msgVersion
msgID
msgMaxSize
msgFlags
msgSecurityModel
msgAuthoritativeEngineID
msgAuthoritativeEngineBoots
msgAuthoritativeEngineTime
msgUserName
msgAuthenticationParameters
msgPrivacyParameters
contextEngineID
contextName
PDU
Generated/processed by Message Processing Model
Generated/Processed by User Security Model (USM)
Scoped PDU (plaintext or encrypted)
scope of authentication
scope of encryption

Figure 5. *SNMPv3 message format with USM.*

MESSAGE PROCESSING AND THE USER SECURITY MODEL

Message processing involves a general-purpose message processing model and a specific security model; this relationship is shown in Fig. 4.

MESSAGE PROCESSING MODEL

RFC 2272 defines a general-purpose message processing model. This model is responsible for accepting PDUs from the Dispatcher, encapsulating them in messages, and invoking the USM to insert security-related parameters in the message header. The message processing model also accepts incoming messages, invokes the USM to process the security-related parameters in the message header, and delivers the encapsulated PDU to the Dispatcher.

Figure 5 illustrates the message structure. The first five fields are generated by the message processing model on outgoing messages and processed by the message processing model on incoming messages. The next six fields show security parameters used by USM. Finally, the PDU, together with the contextEngineID and contextName constitute a scoped PDU, used for PDU processing.

The first five fields are:

msgVersion: Set to snmpv3(3).

msgID: A unique identifier used between two SNMP entities to coordinate request and response messages, and by the message processor to coordinate the processing of the message by different subsystem models within the architecture. The range of this ID is 0 through 2^{31}–1.

msgMaxSize: Conveys the maximum size of a message in octets supported by the sender of the message, with a range of 484 through 2^{31}–1. This is the maximum segment size that the sender can accept from another SNMP engine (whether a response or some other message type).

msgFlags: An octet string containing three flags in the least significant three bits: reportableFlag, privFlag, authFlag. If reportableFlag = 1, then a Report PDU must be returned to the sender under those conditions that can cause the generation of a Report PDU; when the flag is zero, a Report PDU may not be sent. The reportableFlag is set to 1 by the sender in all messages containing a request (Get, Set) or an Inform, and set to 0 for messages containing a Response, a Trap, or a Report PDU. The reportableFlag is a secondary aid in determining when to send a Report. It is only used in cases in which the PDU portion of the message cannot be decoded (e.g., when decryption fails due to incorrect key). The privFlag and authFlag are set by the sender to indicate the security level that was applied to the message. For privFlag = 1, encryption was applied and for privFlag = 0, authentication was applied. All combinations are allowed except (privFlag = 1 AND authFlag = 0); that is, encryption without authentication is not allowed.

msgSecurityModel: An identifier in the range of 0 through 2^{31}–1 that indicates which security model was used by the sender to prepare this message and therefore which security model must be used by the receiver to process this message. Reserved values include 1 for SNMPv1, 2 for SNMPv2c, and 3 for SNMPv3.

USER-BASED SECURITY MODEL

RFC 2274 defines the User Security Model (USM). USM provides authentication and privacy services for SNMP. Specifically, USM is designed to secure against the following principal threats:

- **Modification of Information:** An entity could alter an in-transit message generated by an authorized entity in such a way as to effect unauthorized management operations, including the setting of object values. The essence of this threat is that an unauthorized entity could change any management parameter, including those related to configuration, operations, and accounting.
- **Masquerade:** Management operations that are not authorized for some entity may be attempted by that entity by assuming the identity of an authorized entity.
- **Message Stream Modification:** SNMP is designed to operate over a connectionless transport protocol. There is a threat that SNMP messages could be reordered, delayed, or replayed (duplicated) to effect unauthorized management operations. For example, a message to reboot a device could be copied and replayed later.
- **Disclosure:** An entity could observe exchanges between a manager and an agent and thereby learn the values of managed objects and learn of notifiable events. For example, the observation of a set command that changes passwords would enable an attacker to learn the new passwords.

USM is not intended to secure against the following two threats.

- **Denial of Service:** An attacker may prevent exchanges between a manager and an agent.
- **Traffic Analysis:** An attacker may observe the general pattern of traffic between managers and agents.

The lack of a counter to the denial-of-service threat may be justified on two grounds: First, denial-of-service attacks are in many cases indistinguishable from the type of network failures with which any viable network management application must cope as a matter of course; and second, a denial-of-service attack is likely to disrupt all types of exchanges and is a matter for an overall security facility, not one embedded in a network management protocol. As to traffic analysis, many network management traffic patterns are predictable (e.g., entities may be managed via SNMP commands issued on a regular basis by one or a few management stations) and therefore there is no significant advantage to protecting against observing these traffic patterns.

CRYPTOGRAPHIC FUNCTIONS — Two cryptographic functions are defined for USM: authentication and encryption. To support these functions, an SNMP engine requires two values: a privacy key (privKey) and an authentication key (authKey). Separate values of these two keys are maintained for the following users:

- **Local users:** Any principal at this SNMP engine for which management operations are authorized.
- **Remote users:** Any principal at a remote SNMP engine for which communication is desired.

These values are user attributes stored for each relevant user. The values of privKey and authKey are not accessible via SNMP.

USM allows the use of one of two alternative authentication protocols: HMAC-MD5-96 and HMAC-SHA-96. HMAC uses a secure hash function and a secret key to produce a message authentication code; HMAC is widely used for Internet-based applications and is defined in RFC 2104. For HMAC-MD5-96, HMAC is used with MD5 as the underlying hash function. A 16-octet (128-bit) authKey is used as input to the HMAC algorithm. The algorithm produces a 128-bit output, which is truncated to 12 octets (96 bits). For HMAC-SHA-96, the underling hash function is SHA-1. The authKey is 20 octets in length. The algorithm produces a 20-octet output, which is again truncated to 12 octets.

USM uses the cipher block chaining (CBC) mode of the Data Encryption Standard (DES) for encryption. A 16-octet privKey is provided as input to the encryption protocol. The first eight octets (64 bits) of this privKey are used as a DES key. Because DES only requires a 56-bit key, the least significant bit of each octet is ignored. For CBC mode, a 64-bit initialization vector (IV) is required. The last eight octets of the privKey contain a value that is used to generate this IV.

AUTHORITATIVE AND NON-AUTHORITATIVE ENGINES — In any message transmission, one of the two entities, transmitter or receiver, is designated as the authoritative SNMP engine, according to the following rules:

- When an SNMP message contains a payload which expects a response (for example, a Get, GetNext, GetBulk, Set, or Inform PDU), then the receiver of such messages is authoritative.
- When an SNMP message contains a payload which does not expect a response (for example, an SNMPv2-Trap, Response, or Report PDU), then the sender of such a message is authoritative.

Thus, for messages sent on behalf of a Command Generator and for Inform messages from a Notification Originator, the receiver is authoritative. For messages sent on behalf of a Command Responder or for Trap messages from a Notification Originator, the sender is authoritative. This designation serves two purposes:

1. The timeliness of a message is determined with respect to a clock maintained by the authoritative engine. When an authoritative engine sends a message (Trap, Response, Report), it contains the current value of its clock, so that the non-authoritative recipient can synchronize on that clock. When a non-authoritative engine sends a message (Get, GetNext, GetBulk, Set, Inform), it includes its current estimate of the time value at the destination, allowing the destination to assess the message's timeliness.
2. A key localization process, described later, enables a single principal to own keys stored in multiple engines; these keys are localized to the authoritative engine in such a way that the principal is responsible for a single key but avoids the security risk of storing multiple copies of the same key in a distributed network.

It makes sense to designate the receiver of Command Generator and Inform PDUs as the authoritative engine, and therefore responsible for checking message timeliness. If a response or trap is delayed or replayed, little harm should occur. However, Command Generator and, to some extent, Inform PDUs result in management operations, such as reading or setting MIB objects. Thus, it is important to guarantee that such PDUs are not delayed or replayed, which could cause undesired effects.

USM MESSAGE PARAMETERS — When an outgoing message is passed to the USM by the Message Processor, the USM fills in the security-related parameters in the message header. When an incoming message is passed to the USM by the Message Processor, the USM processes the values contained in those fields. The security-related parameters are the following:

- **msgAuthoritativeEngineID:** The snmpEngineID of the authoritative SNMP engine involved in the exchange of this message. Thus, this value refers to the source for a

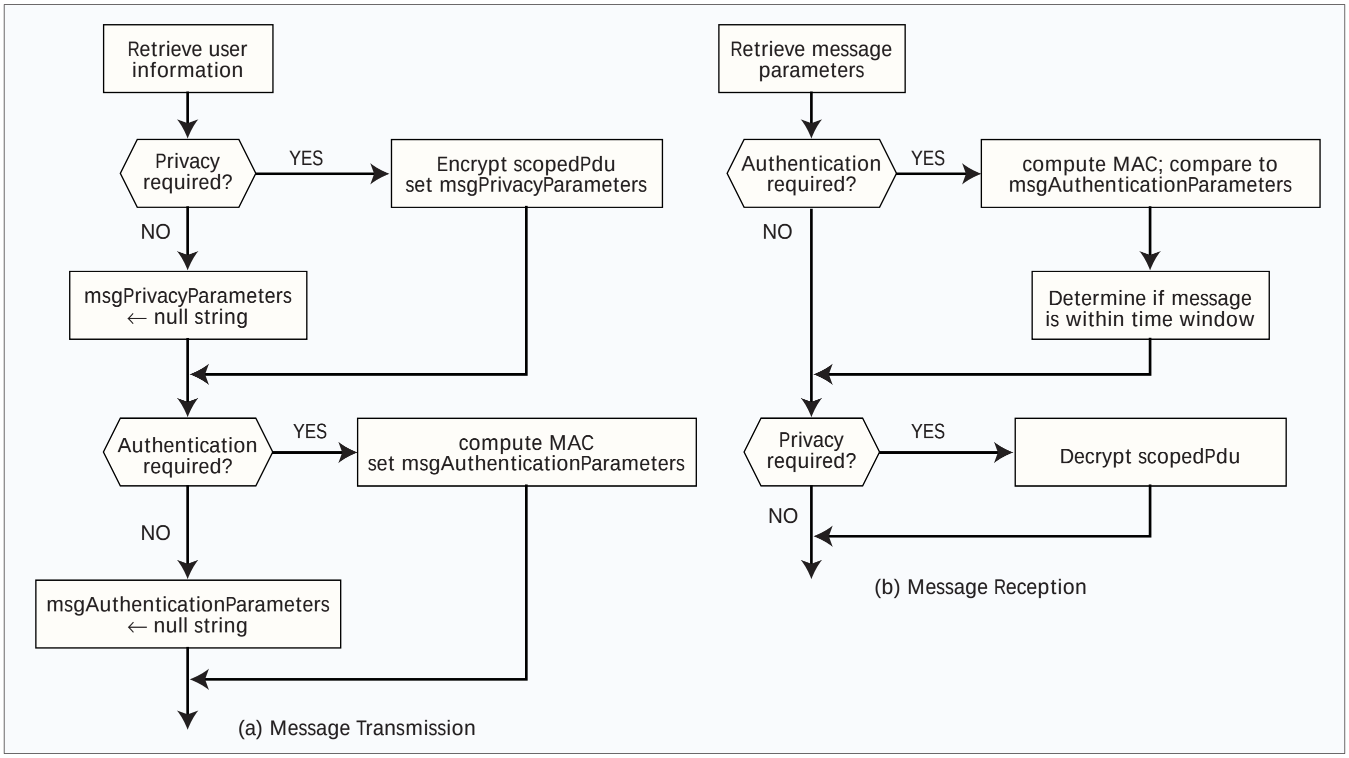
Trap, Response, or Report, and to the destination for a Get, GetNext, GetBulk, Set, or Inform.

- **msgAuthoritativeEngineBoots:** The snmpEngineBoots value of the authoritative SNMP engine involved in the exchange of this message. The object snmpEngineBoots is an integer in the range 0 through $2^{31} - 1$ that represents the number of times that this SNMP engine has initialized or reinitialized itself since its initial configuration.
- **msgAuthoritativeEngineTime:** The snmpEngineTime value of the authoritative SNMP engine involved in the exchange of this message. The object snmpEngineTime is an integer in the range 0 through $2^{31} - 1$ that represents the number of seconds since this authoritative SNMP engine last incremented the snmpEngineBoots object. Each authoritative SNMP engine is responsible for incrementing its own snmpEngineTime value once per second. A non-authoritative engine is responsible for incrementing its notion of snmpEngineTime for each remote authoritative engine with which it communicates.
- **msgUserName:** The user (principal) on whose behalf the message is being exchanged.
- **msgAuthenticationParameters:** Null if authentication is not being used for this exchange. Otherwise, this is an authentication parameter. For the current definition of USM, the authentication parameter is an HMAC message authentication code.
- **msgPrivacyParameters:** Null if privacy is not being used for this exchange. Otherwise, this is a privacy parameter. For the current definition of USM, the privacy parameter is a value used to form the initial value (IV) in the DES CBC algorithm.

Figure 6 summarizes the operation of USM. For message transmission, encryption is performed first, if needed. The scoped PDU is encrypted and placed in the message payload, and the msgPrivacyParameters value is set to the value needed to generate the IV. Then, authentication is performed, if needed. The entire message, including the scoped PDU is input to HMAC, and the resulting authentication code is placed in msgAuthenticationParameters. For incoming messages, authentication is performed first if needed. USM first checks the incoming MAC against a MAC that it calculated; if the two values match, then the message is assumed to be authentic (comes from the alleged source and has not been altered in transmission). Then, USM checks whether the message is within a valid time window, as explained below. If the message is not timely, it is discarded as not authentic. Finally, if the scoped PDU has been encrypted, USM performs a decryption and returns the plaintext.

USM Timeliness Mechanisms — USM includes a set of timeliness mechanisms to guard against message delay and message replay. Each SNMP engine that can ever act in the capacity of an authoritative engine must maintain two objects, snmpEngineBoots and snmpEngineTime, that refer to its local time. When an SNMP engine is first installed, these two object values are set to 0. Thereafter, snmpEngineTime is incremented once per second. If snmpEngineTime ever reaches its maximum value ($2^{31} - 1$), snmpEngineBoots is incremented, as if the system had rebooted, and snmpEngineTime is set to 0 and begins incrementing again. Using a synchronization mechanism, a non-authoritative engine maintains an estimate of the time values for each authoritative engine with which it communicates. These estimated values are placed in each outgoing message, and enable the receiving authoritative engine to determine whether or not the incoming message is timely.

The synchronization mechanism works in the following fashion. A non-authoritative engine keeps a local copy of three variables for each authoritative SNMP engine that is known to this engine:


Retrieve user information
Privacy required?
YES
Encrypt scopedPdu set msgPrivacyParameters
NO
msgPrivacyParameters ← null string
Authentication required?
YES
compute MAC set msgAuthenticationParameters
NO
msgAuthenticationParameters ← null string
(a) Message Transmission
Retrieve message parameters
Authentication required?
YES
compute MAC; compare to msgAuthenticationParameters
NO
Determine if message is within time window
Privacy required?
YES
Decrypt scopedPdu
NO
(b) Message Reception

■ **Figure 6.** *USM message processing.*

- snmpEngineBoots: the most recent value of snmpEngineBoots for the remote authoritative engine.
- snmpEngineTime: this engine's notion of snmpEngineTime for the remote authoritative engine. This value is synchronized to the remote authoritative engine by the synchronization process described below. Between synchronization events, this value is logically incremented once per second to maintain a loose synchronization with the remote authoritative engine.
- latestReceivedEngineTime: the highest value of msgAuthoritativeEngineTime that has been received by the this engine from the remote authoritative engine; this value is updated whenever a larger value of msgAuthoritativeEngineTime is received. The purpose of this variable is to protect against a replay message attack that would prevent the non-authoritative SNMP engine's notion of snmpEngineTime from advancing.

One set of these three variables is maintained for each remote authoritative engine known to this engine. Logically, the values are maintained in some sort of cache, indexed by the unique snmpEngineID of each remote authoritative engine.

To enable non-authoritative engines to maintain time synchronization, each authoritative engine inserts its current boot and time values, as well as its value of snmpEngineID, in each outgoing Response, Report, or Trap message in the fields msgAuthoritativeEngineBoots, msgAuthoritativeEngineTime, and msgAuthoritativeEngineID. If the message is authentic, and if the message is within the time window, then the receiving non-authoritative engine updates is local variables (snmpEngineBoots, snmpEngineTime, and latestReceived EngineTime) for that remote engine according to the following rules:

1. An update occurs if at least one of the two following conditions is true:
- (msgAuthoritativeEngineBoots > snmpEngineBoots) or
- [(msgAuthoritativeEngineBoots = snmpEngineBoots) and (msgAuthoritativeEngineTime > latestReceivedEngineTime)]

The first condition says that an update should occur if the boot value from the authoritative engine has increased since the last update. The second condition says that if the boot value has not increased, then an update should occur if the incoming engine time is greater than the latest received engine time. The incoming engine time will be less than the latest received engine time if two incoming messages arrive out of order, which can happen, or if a replay attack is underway; in either case, the receiving engine will not perform an update.

2. If an update is called for, then the following changes are made:
- set snmpEngineBoots to the value of msgAuthoritativeEngineBoots
- set snmpEngineTime to the value of msgAuthoritativeEngineTime
- set latestReceivedEngineTime to the value of msgAuthoritativeEngineTime

If we turn the logic around, we see that if msgAuthoritativeEngineBoots < snmpEngineBoots, then no update occurs. Such a message is considered not authentic and must be ignored. If msgAuthoritativeEngineBoots = snmpEngineBoots but msgAuthoritativeEngineTime < latestReceivedEngineTime, then again no update occurs. In this case, the message may be authentic, but it may be misordered, in which case an update of snmpEngineTime is not warranted.

Note that synchronization is only employed if the authentication service is in use for this message and the message has been determined to be authentic via HMAC. This restriction is essential because the scope of authentication includes msgAuthoritativeEngineID, msgAuthoritativeEngineBoots, and msgAuthoritativeEngineTime, thus assuring that these values are valid.

SNMPv3 dictates that a message must be received within a reasonable time window, to avoid delay and replay attacks. The time window should be chosen to be as small as possible given the accuracy of the clocks involved, round-trip communication delays, and the frequency with which clocks are synchronized. If the time window is set too small, authentic messages will be rejected as unauthentic. On the other hand, a large time window increases the vulnerability to malicious delays of messages.

We consider the more important case of an authoritative receiver; timeliness testing by a non-authoritative receiver differs slightly. With each incoming message that has been authenticated and whose msgAuthoritativeEngineID is the same as the value of snmpEngineID for this engine, the engine compares the values of msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime from the incoming message with the values of snmpEngineBoots and snmpEngineTime that this engine maintains for itself. The incoming message is considered outside the time window if any one of the following conditions is true:

- snmpEngineBoots = $2^{31} - 1$ or
- msgAuthoritativeEngineBoots $\neq$ snmpEngineBoots or
- the value of msgAuthoritativeEngineTime differs from that of snmpEngineTime by more than ±150 s

The first condition says that if snmpEngineBoots is latched at its maximum value, no incoming message can be considered authentic. The second condition says that a message must have a boot time equal to the local engine's boot time; for example, if the local engine has rebooted and the remote engine has not synchronized with the local engine since the reboot, the messages from that remote engine are considered not authentic. The final condition says that the time on the incoming message must be greater than the local time minus 150 s and less than the local time plus 150 s.

If a message is considered to be outside the time window, then the message is considered not authentic, and an error indication (notInTimeWindow) is returned to the calling module.

Again, as with synchronization, timeliness checking is only done if the authentication service is in use and the message is authentic, assuring the validity of the message header fields.

KEY LOCALIZATION — A requirement for the use of the authentication and privacy services of SNMPv3 is that, for any communication between a principal on a non-authoritative engine and a remote authoritative engine, a secret authentication key and a secret privacy key must be shared. These keys enable a user at a non-authoritative engine (typically a management system) to employ authentication and privacy with remote authoritative systems that the user manages (typically, agent systems). RFC 2274 provides guidelines for the creation, update, and management of these keys.

To simplify the key management burden on principals, each principal is only required to maintain a single authentication key and a single encryption key. These keys are not stored in a MIB and are not accessible via SNMP. In this sec-

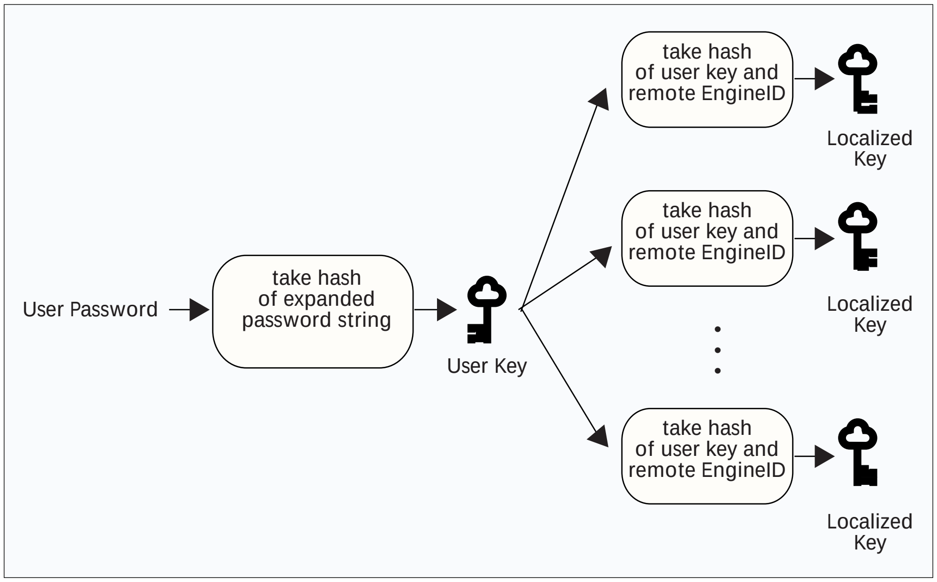


Figure 7. *Key localization.*

tion we look first at the technique for generating these keys from a password. Then we look at the concept of key localization, which enables a principal to share a unique authentication and encryption key with each remote engine while only maintaining a single authentication and encryption key locally. These two techniques were first proposed in [2].

A user requires a 16-octet privacy key and an authentication key of length either 16 or 20 octets. For keys owned by human users, it is desirable that the user be able to employ a human-readable password rather than a bit-string key. Accordingly, RFC 2274 defines an algorithm for mapping from the user password to a 16- or 20-octet key. USM places no restriction on the password itself, but local management policies should dictate that users employ passwords that are not easily guessed.

Password to key generation is performed as follows:

1. Take the user's password as input and produce a string of length 2^{20} octets (1,048,576 octets) by repeating the password value as many times as necessary, truncating the last value if necessary, to form the string digest0. For example, an eight-character password (23 octets) would be concatenated with itself 2^{17} times to form digest0.
2. If a 16-octet key is desired, take the MD5 hash of digest0 to form digest1. If a 20-octet key is desired, take the SHA-1 hash of digest0 to form digest1. The output is the user's key.

One advantage of this technique is that it greatly slows down a brute-force dictionary attack, in which an adversary tries many different potential passwords, generating the key from each one, and then tests whether the resulting key works with the authentication or encryption data available to him or her. For example, if an attacker intercepts an authenticated message, the attacker could try generating the HMAC value with different possible user keys. If a match occurs, the attacker can assume that the password has been discovered. The two-step process outlined above significantly increases the amount of time such an attack will take.

Another advantage of this technique is that it decouples the user's keys from any particular network management system (NMS). No NMS need store values of user keys. Instead, when needed, a user key is generated from that user's password. Reference [2] lists the following considerations that motivate the use of a password approach that is independent of NMS:

- If a key is to be stored rather than generated from a password, one alternative is to maintain a centralized repository of secret keys. But this adversely affects overall reliability and can make troubleshooting impossible if the repository itself is not accessible when needed.
- On the other hand, if duplicate repositories are maintained, this endangers overall security by providing potential breakers with more targets to attack.
- If either centralized or multiple duplicate repositories are used, they must be maintained in secure locations. This may reduce the opportunity for "forward camp" establishment during firefighting (i.e., troubleshooting when unpredictable segments of the network are inoperative and/or inaccessible for an unpredictable length of time).

A single password could be used to generate a single key for both authentication and encryption. A more secure scheme is to use two passwords, one to generate an authentication key and one to generate a distinct encryption key.

A localized key is defined in RFC 2274 as a secret key shared between a user and one authoritative SNMP engine. The objective is that the user need only maintain a single key (or two keys of both authentication and privacy are required) and therefore need only remember one password (or two). The actual secrets shared between a particular user and each authoritative SNMP engine are different. The process by which a single user key is converted into multiple unique keys, one for each remote SNMP engine, is referred to as key localization. Reference [2] gives a motivation for this strategy, which we summarize here.

We can define the following goals for key management:

- Every SNMP agent system in a distributed network has its own unique key for every user authorized to manage it. If multiple users are authorized as managers, the agent has a unique authentication key and a unique encryption key for each user. Thus, if the key for one user is compromised, the keys for other users are not compromised.
- The keys for a user on different agents are different. Thus, if an agent is compromised, only the user keys for that agent are compromised and not the user keys in use for other agents.
- Network management can be performed from any point on the network, regardless of the availability of a preconfigured network management system (NMS). This allows a user to perform management functions from any management station. This capability is provided by the password-to-key algorithm described previously.

We can also define the following as things to avoid:

- A user has to remember (or otherwise manage) a large number of keys, a number that grows with the addition of new managed agents.
- An adversary who learns a key for one agent is now able to impersonate any other agent to any user, or any user to any other agent.

To address the preceding goals and considerations, a single user key is mapped by means of a nonreversible one-way function (i.e., a secure hash function) into different localized keys for different authenticated engines (different agents). The procedure is as follows:

1. Form the string digest2 by concatenating digest1 (described above), the authoritative engine's snmpEngineID value, and digest1.
2. If a 16-octet key is desired, take the MD5 hash of digest2. If a 20-octet key is desired, take the SHA-1 hash of digest2. The output is the user's localized key.

The resulting localized key can then be configured on the agent's system in some secure fashion. Because of the one-way nature of MD5 and SHA-1, it is infeasible for an adversary to learn a user key even if the adversary manages to discover a localized key.

Figure 7 summarizes the key localization process.

VIEW-BASED ACCESS CONTROL

Access control is a security function performed at the PDU level. An access control document defines mechanisms for determining whether access to a managed object in a local MIB by a remote principal should be allowed. Conceivably, multiple access control mechanisms could be defined. The SNMPv3 documents define the view-based access control (VACM) model.

VACM has two important characteristics:

- VACM determines whether access to a managed object in a local MIB by a remote principal should be allowed.
- VACM makes use of a MIB that:
  Defines the access control policy for this agent
  Makes it possible for remote configuration to be used.

ELEMENTS OF THE VACM MODEL

RFC 2275 defines five elements that make up the VACM: groups, security level, contexts, MIB views, and access policy.

GROUPS — A group is defined as a set of zero or more <securityModel, securityName> tuples on whose behalf SNMP management objects can be accessed. A securityName refers to a principal, and access rights for all principals in a given group are identical. A unique groupName is associated with each group. The group concept is a useful tool for categorizing managers with respect to access rights. For example, all top-level managers may have one set of access rights, while intermediate-level managers may have a different set of access rights.

Any given combination of securityModel and securityName can belong to at most one group. That is, for this agent, a given principal whose communications are protected by a given securityModel can only be included in one group.

SECURITY LEVEL — The access rights for a group may differ depending on the security level of the message that contains the request. For example, an agent may allow read-only access for a request communicated in an unauthenticated message but may require authentication for write access. Further, for certain sensitive objects, the agent may require that the request and its response be communicated using the privacy service.

CONTEXTS — A MIB context is a named subset of the object instances in the local MIB. Contexts provide a useful way of aggregating objects into collections with different access policies.

The context is a concept that relates to access control. When a management station interacts with an agent to access management information at the agent, then the interaction is between a management principal and the agent's SNMP engine, and the access control privileges are expressed in a MIB view that applies to this principal and this context. Contexts have the following key characteristics:

- An SNMP entity, uniquely identified by a contextEngineID, may maintain more than one context.
- An object or an object instance may appear in more than one context.
- When multiple contexts exist, to identify an individual object instance, its contextName and contextEngineID must be identified in addition to its object type and its instance.

MIB VIEWS — It is often the case that we would like to restrict the access of a particular group to a subset of the managed objects at an agent. To achieve this objective, access to a context is by means of a MIB view, which defines a specific set of managed objects (and optionally specific object instances). VACM makes use of a powerful and flexible technique for defining MIB views, based on the concepts of view subtrees and view families. The MIB view is defined in terms of a collection, or family, of subtrees, with each subtree being included in or excluded from the view.

The managed objects in a local database are organized into a hierarchy, or tree, based on the object identifiers of the objects. This local database comprises a subset of all object types defined according to the Internet-standard Structure of Management Information (SMI) and includes object instances whose identifiers conform to the SMI conventions.

SNMPv3 includes the concept of a subtree. A subtree is simply a node in the MIB's naming hierarchy plus all of its subordinate elements. More formally, a subtree may be defined as the set of all objects and object instances that have a common ASN.1 OBJECT IDENTIFIER prefix to their names. The longest common prefix of all of the instances in the subtree is the object identifier of the parent node of that subtree.

Associated with each entry in vacmAccessTable are three MIB views, one each for read, write, and notify access. Each MIB view consists of a set of view subtrees. Each view subtree in the MIB view is specified as being included or excluded. That is, the MIB view either includes or excludes all object instances contained in that subtree. In addition, a view mask is defined in order to reduce the amount of configuration information required when fine-grained access control is required (e.g., access control at the object instance level).

ACCESS POLICY — VACM enables an SNMP engine to be configured to enforce a particular set of access rights. Access determination depends on the following factors:

- The **principal** making the access request. The VACM makes it possible for an agent to allow different access privileges for different users. For example, a manager system responsible for network-wide configuration may have broad authority to alter items in the local MIB, while an intermediate level manager with monitoring responsibility may have read-only access and may further be limited to accessing only a subset of the local MIB. As was discussed, principals are assigned to groups and access policy is specified with respect to groups.
- The **security level** by which the request was communicated in an SNMP message. Typically, an agent will require the use of authentication for messages containing a set request (write operation).

- The **security model** used for processing the request message. If multiple security models are implemented at an agent, the agent may be configured to provide different levels of access to requests communicated by messages processed by different security models. For example, certain items may be accessible if the request message comes through USM, but not accessible if the Security Model is SNMPv1.
- The **MIB context** for the request.
- The specific **object instance** for which access is requested. Some objects hold more critical or sensitive information than others, and therefore the access policy must depend on the specific object instance requested.
- The **type of access** requested (read, write, notify). Read, write, and notify are distinct management operations, and different access control policies may apply for each of these operations.

Access Control Processing

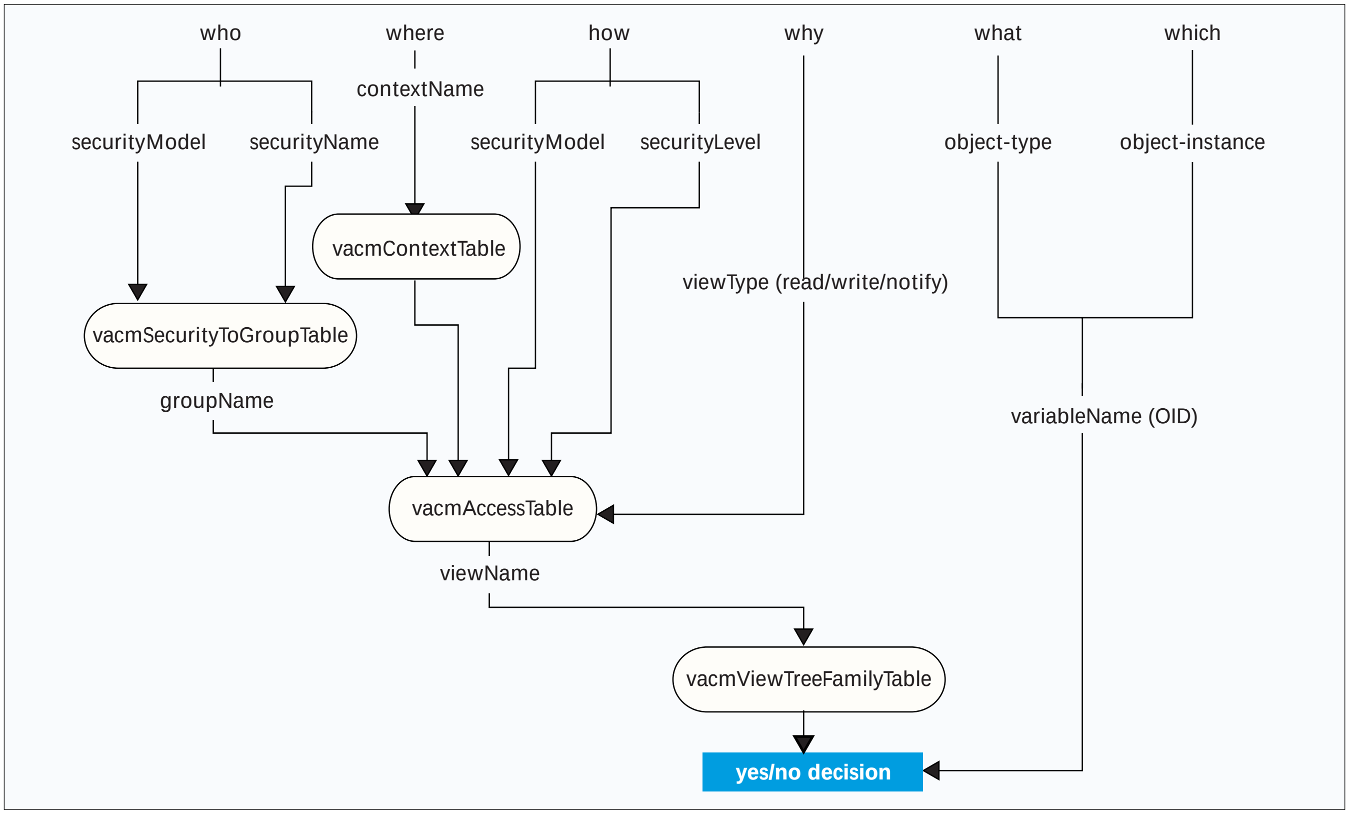
An SNMP application invokes VACM via the isAccessAllowed primitive, with the input parameters securityModel, securityName, securityLevel, viewType, contextName, and variableName. All of these parameters are needed to make the access control decision. Put another way, the Access Control Subsystem is defined in such a way as to provide a very flexible tool for configuring access control at the agent, by breaking down the components of the access control decision into six separate variables.

Figure 8, adapted from a figure in RFC 2275, provides a useful way of looking at the input variables and shows how the various tables in the VACM MIB come into play in making the access control decision.

- **who:** The combination of securityModel and securityName define the who of this operation; it identifies a given principal whose communications are protected by a given securityModel. This combination belongs to at most one group at this SNMP engine. The vacmSecurityToGroupTable provides the groupName, given the securityModel and securityName.
- **where:** The contextName specifies where the desired management object is to be found. The vacmContextTable contains a list of the recognized contextNames.
- **how:** The combination of securityModel and securityLevel defines how the incoming request or Inform PDU was protected. The combination of who, where, and how identifies zero or one entries in vacmAccessTable.
- **why:** The viewType specifies why access is requested: for a read, write, or notify operation. The selected entry in vacmAccessTable contains one MIB viewName for each of these three types of operation, and viewType is used to select a specific viewName. This viewName selects the appropriate MIB view from vacmViewTreeFamilyTable.
- **what:** The variableName is an object identifier whose prefix identifies a specific object type and whose suffix identifies a specific object instance. The object type indicates what type of management information is requested.
- **which:** The object instance indicates which specific item of information is requested.

Finally, the variableName is compared to the retrieved MIB view. If the variableName matches an element included in the MIB view, then access is granted.

Motivation — The concepts that make up VACM appear to result in a rather complex definition of access control The motivations for introducing these concepts are to clarify the relationships involved in accessing management



■ **Figure 8.** *VACM logic.*

information and to minimize the storage and processing requirements at the agent. To understand these motivations, consider the following. In SNMPv1 and SNMPv2c, the community concept is used to represent the following security-related information:

- The identity of the requesting entity (management station)
- The identity of the performing entity (agent acting for itself or for a proxied entity)
- The identity of the location of the management information to be accessed (agent or proxied entity)
- Authentication information
- Access control information (authorization to perform required operation)
- MIB view information

By lumping all of these concepts into a single variable, flexibility and functionality are lost. VACM provides the same set of security-related information using distinct variables for each item. This is a substantial improvement over SNMPv1. It uncouples various concepts so that values can be assigned to each one separately.

SUMMARY

SNMPv2 was a substantial improvement over SNMPv1, while retaining its essential character of ease of understanding and ease of implementation. Version 2 provides better support for a decentralized network management architecture, enhances performance, and provides a few other bells and whistles of interest to application developers.

SNMPv3 fixes the most obvious failing of versions 1 and 2: lack of security. The security enhancements to SNMP are reasonably simple and straightforward. They provide the key security features missing from SNMP: privacy, authentication, and access control. There is now, at last, a worthy successor to SNMPv1, and the new standard should succeed in the marketplace. Vendors are likely to adopt the new version to provide more features and more efficient operation to their users. And, we can expect additional MIBs to be defined within the SNMPv3 framework to extend its scope of support for various network management applications.

FURTHER READING

Two books provide more in-depth coverage of SNMPv3 [3, 4]. These Web sites also provide useful information:

- http://ietf.org/html.charters/snmpv3-charter.html — Home page of the SNMPv3 working group. This site maintains the most recent copies of SNMPv3-related RFCs and internet draft documents, plus a schedule of past and future work.
- http://www.ibr.cs.tu-bs.de/projects/snmpv3 — An SNMPv3 Web site maintained by the Technical University of Braunschweig. It provides links to the RFCs and internet drafts, copies of clarifications and proposed changes posted by the working group, and links to vendors with SNMPv3 implementations.
- http://www.simple-times.org — *The Simple Times* is an online free quarterly publication of SNMP technology, comment, and events. The Web site is maintained jointly by the Technical University of Braunschweig and the University of Twente. Each issue features technical articles, columns, a list of Internet documents, and brief announcements of events and products.
- http://wwwsnmp.cs.utwente.nl — This is known as The Simple Web site, maintained by the University of Twente. It is a good source of information on SNMP, including pointers to many public-domain implementations and lists of books and articles.

REFERENCES

[1] Stallings, W. "SNMP and SNMPv2: The Infrastructure for Network Management." *IEEE Commun. Mag.,* March 1998..

[2] Blumenthal, U., Hien, N., and Wijnen, B. "Key Derivation for Network Management Applications." *IEEE Network,* May/June, 1997.

[3] Blumenthal, U.; Hien, N.; and Wijnen, B. SNMPv3 Handbook. Reading, MA: Addison-Wesley, 1999.

[4] Stallings, W. SNMP, SNMPv2, SNMPv3, and RMON 1 and 2, Third Edition. Reading, MA: Addison-Wesley, 1998.

BIOGRAPHY

WILLIAM STALLINGS (ws@shore.net) is a consultant, lecturer, and author of over a dozen professional reference books and textbooks on data communications and computer networking. He has three times received the award for the best Computer Science textbook of the year from the Text and Academic Authors Association (1998: *Operating Systems,* 3rd ed.; 1997: *Data and Computer Communications,* 5th ed.; 1996: *Computer Organization and Architecture,* 4th ed.). He has a Ph.D. from M.I.T. in computer science. His home in cyberspace is http://www.shore.net/~ws.

APPENDIX: OVERVIEW OF CRYPTOGRAPHIC FUNCTIONS

The principal cryptographic algorithms used in SNMPv3, are a conventional encryption algorithm, a secure hash function, and a message authentication code. These three types of algorithms are briefly reviewed in this appendix.

CONVENTIONAL ENCRYPTION

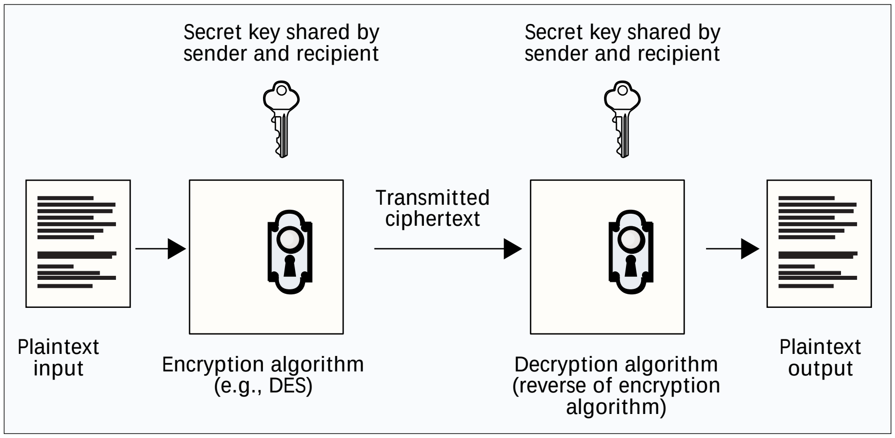
Conventional encryption, also referred to as symmetric encryption or single-key encryption, was the only type of encryption in use prior to the introduction of public-key encryption in the late 1970s. Conventional encryption has been used for secret communication by countless individuals and groups, from Julius Caesar to the German U-boat force to present-day diplomatic, military, and commercial users. It remains by far the more widely used of the two types of encryption.

A conventional encryption scheme has five ingredients (Fig. 9):

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the algorithm. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the same secret key and produces the original plaintext.

There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or


Secret key shared by sender and recipient
Secret key shared by sender and recipient
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

■ **Figure 9.** *Conventional encryption.*

figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.

2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

There are two general approaches to attacking a conventional encryption scheme. The first attack is known as cryptanalysis. Cryptanalytic attacks rely on the nature of the algorithm plus (perhaps) some knowledge of the general characteristics of the plaintext or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used. Of course, if the attack succeeds in deducing the key, the effect is catastrophic: all future and past messages encrypted with that key are compromised.

The second method, known as the brute-force attack, is to try every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

SECURE HASH FUNCTION

An essential element of most authentication and digital signature schemes is a secure hash function. A hash function accepts a variable-length message M as input and produces a fixed size hash code H(M), sometimes called a message digest, as output. A hash value is generated by a function H of the form

h = H(M)

where M is a variable-length message, and H(M) is the fixed-length hash value.

To be useful for message authentication and digital signature, a hash function H must have the following properties:

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given code m, it is computationally infeasible to find x such that H(x) = m.
5. For any given block x, it is computationally infeasible to find y not equal to x such that H(y) = H(x).
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).

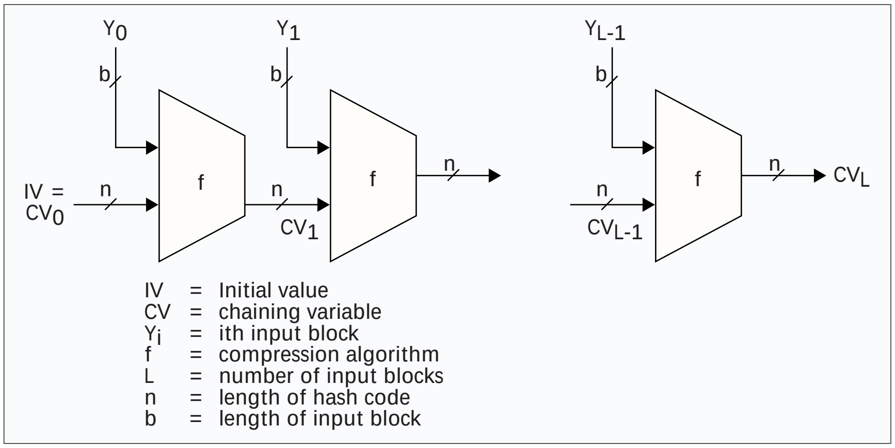
The first three properties are requirements for the practical application of a hash function to message authentication. The fourth property is a "one-way" property: it is easy to generate a code given a message but virtually impossible to generate a message given a code. The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found. The sixth property protects against a sophisticated class of attack known as the birthday attack.

The overall structure of a typical secure hash function is illustrated in Fig. 10. This structure, referred to as an iterated hash function, is the structure of most hash functions in use today. The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each. If necessary, the final block is padded to b bits. The final block also includes the value of the total length of the input to the hash function. The inclusion of the length makes the job of the opponent more difficult. Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value.

The hash algorithm involves repeated use of a *compression function*, f, that takes two inputs (an n-bit input from the previous step, called the *chaining variable*, and a b-bit block) and produces an n-bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Usually, $b > n$; hence the term compression. The hash function can be summarized as follows:

$$\begin{aligned} CV_0 &= IV = \text{initial } n\text{-bit value} \\ CV_i &= f(CV_{i-1}, Y_{i-1}) \quad 1 \le i \le L \\ H(M) &= CV_L \end{aligned}$$

where input to the hash function is a message M consisting of the blocks Y_0, Y_1, … , Y_{L-1}.



■ **Figure 10.** *General structure of secure Hash code.*

Message Authentication Code

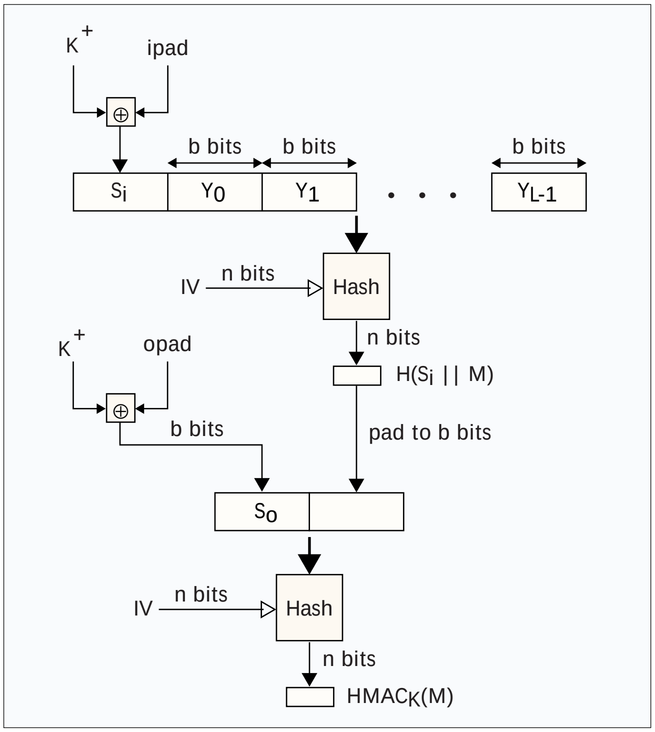
Message authentication is a procedure that allows communicating parties to verify that received messages are authentic. The two important aspects are to verify that the contents of the message have not been altered and that the source is authentic. The message authentication code (MAC) is a widely-used technique for performing message authentication, and one MAC algorithm has emerged as the Internet standard for a wide variety of applications: HMAC.

A MAC algorithm involves the use of a secret key to generate a small block of data, known as a message authentication code, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key K. When A has a message to send to B, it calculates the message authentication code as a function of the message and the key. The message plus code are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new message authentication code. The received code is compared to the calculated code. If we assume that only the receiver and the sender know the identity of the secret key, and if the received code matches the calculated code, then:

- The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the code, then the receiver's calculation of the code will differ from the received code. Because the attacker is assumed not to know the secret key, the attacker can not alter the code to correspond to the alterations in the message.
- The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper code.
- If the message includes a sequence number (such as is used with X.25, HDLC, and TCP), then the receiver can be assured of the proper sequence, because an attacker can not successfully alter the sequence number.

Figure 11 illustrates the overall operation of HMAC. Define the following terms:

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
M = message input to HMAC (including the padding specified in the embedded hash function)
Y_i = ith block of M, $0 \le i \le (L-1)$
L = number of blocks in M
b = number of bits in a block
n = length of hash code produced by embedded hash function
K = secret key; if key length is greater than b, the key is input to the hash function to produce an n-bit key; recommended length is $\ge n$



■ **Figure 11.** *HMAC structure.*

K^+ = K padded with zeros on the left so that the result is b bits in length
ipad = 00110110 (36 in hexadecimal) repeated b/8 times
opad = 01011100 (5C in hexadecimal) repeated b/8 times

Then HMAC can be expressed as follows:

$$\mathrm{HMAC}_K = \mathrm{H}\left[\left(K^+ \oplus \mathrm{opad}\right) \| \mathrm{H}\left[\left(K^+ \oplus \mathrm{ipad}\right) \| \mathrm{M}\right]\right]$$

In words:

1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and b = 512, then K will be appended with 44 zero bytes 0 x 00).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o.
6. Append the hash result from step 4 to S_o.
7. Apply H to the stream generated in step 6 and output the result.

Note that the XOR with ipad results in flipping one-half of the bits of K. Similarly, the XOR with opad results in flipping one-half of the bits of K, but a different set of bits. In effect, by passing S_i and S_o through the hash algorithm, we have pseudorandomly generated two keys from K.