            User-based Security Model (USM) for version 3 of the
                Simple Network Management Protocol (SNMPv3)

Status of this Memo

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

Copyright Notice

Abstract

   This document describes the User-based Security Model (USM) for SNMP
   version 3 for use in the SNMP architecture [RFC2571]. It defines the
   Elements of Procedure for providing SNMP message level security.
   This document also includes a MIB for remotely monitoring/managing
   the configuration parameters for this Security Model.

Table of Contents

1. Introduction

    The Architecture for describing Internet Management Frameworks
    [RFC2571] describes that an SNMP engine is composed of:

      1) a Dispatcher
      2) a Message Processing Subsystem,
      3) a Security Subsystem, and
      4) an Access Control Subsystem.

    Applications make use of the services of these subsystems.

    It is important to understand the SNMP architecture and the
    terminology of the architecture to understand where the Security
    Model described in this document fits into the architecture and

interacts with other subsystems within the architecture. The reader
is expected to have read and understood the description of the SNMP
architecture, as defined in [RFC2571].

This memo [RFC2274] describes the User-based Security Model as it is
used within the SNMP Architecture. The main idea is that we use the
traditional concept of a user (identified by a userName) with which
to associate security information.

This memo describes the use of HMAC-MD5-96 and HMAC-SHA-96 as the
authentication protocols and the use of CBC-DES as the privacy
protocol. The User-based Security Model however allows for other such
protocols to be used instead of or concurrent with these protocols.
Therefore, the description of HMAC-MD5-96, HMAC-SHA-96 and CBC-DES
are in separate sections to reflect their self-contained nature and
to indicate that they can be replaced or supplemented in the future.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [RFC2119].

1.1. Threats

Several of the classical threats to network protocols are applicable
to the network management problem and therefore would be applicable
to any SNMP Security Model. Other threats are not applicable to the
network management problem. This section discusses principal
threats, secondary threats, and threats which are of lesser
importance.

The principal threats against which this SNMP Security Model should
provide protection are:

- Modification of Information
  The modification threat is the danger that some unauthorized entity
  may alter in-transit SNMP messages generated on behalf of an
  authorized principal in such a way as to effect unauthorized
  management operations, including falsifying the value of an object.

- Masquerade
  The masquerade threat is the danger that management operations not
  authorized for some user may be attempted by assuming the identity
  of another user that has the appropriate authorizations.

Two secondary threats are also identified. The Security Model
defined in this memo provides limited protection against:

    - Disclosure
      The disclosure threat is the danger of eavesdropping on the
      exchanges between managed agents and a management station.
      Protecting against this threat may be required as a matter of local
      policy.

    - Message Stream Modification
      The SNMP protocol is typically based upon a connection-less
      transport service which may operate over any sub-network service.
      The re-ordering, delay or replay of messages can and does occur
      through the natural operation of many such sub-network services.
      The message stream modification threat is the danger that messages
      may be maliciously re-ordered, delayed or replayed to an extent
      which is greater than can occur through the natural operation of a
      sub-network service, in order to effect unauthorized management
      operations.

   There are at least two threats that an SNMP Security Model need not
   protect against. The security protocols defined in this memo do not
   provide protection against:

   - Denial of Service
     This SNMP Security Model does not attempt to address the broad
     range of attacks by which service on behalf of authorized users is
     denied. Indeed, such denial-of-service attacks are in many cases
     indistinguishable from the type of network failures with which any
     viable network management protocol must cope as a matter of course.
   - Traffic Analysis
     This SNMP Security Model does not attempt to address traffic
     analysis attacks. Indeed, many traffic patterns are predictable -
     devices may be managed on a regular basis by a relatively small
     number of management applications - and therefore there is no
     significant advantage afforded by protecting against traffic
     analysis.

1.2. Goals and Constraints

   Based on the foregoing account of threats in the SNMP network
   management environment, the goals of this SNMP Security Model are as
   follows.

   1) Provide for verification that each received SNMP message has
      not been modified during its transmission through the network.

   2) Provide for verification of the identity of the user on whose
      behalf a received SNMP message claims to have been generated.

   3) Provide for detection of received SNMP messages, which request
      or contain management information, whose time of generation was
      not recent.

   4) Provide, when necessary, that the contents of each received
      SNMP message are protected from disclosure.

   In addition to the principal goal of supporting secure network
   management, the design of this SNMP Security Model is also influenced
   by the following constraints:

   1) When the requirements of effective management in times of
      network stress are inconsistent with those of security, the design
      should prefer the former.

   2) Neither the security protocol nor its underlying security
      mechanisms should depend upon the ready availability of other
      network services (e.g., Network Time Protocol (NTP) or key
      management protocols).

   3) A security mechanism should entail no changes to the basic
      SNMP network management philosophy.

1.3. Security Services

   The security services necessary to support the goals of this SNMP
   Security Model are as follows:

   - Data Integrity
     is the provision of the property that data has not been altered or
     destroyed in an unauthorized manner, nor have data sequences been
     altered to an extent greater than can occur non-maliciously.

   - Data Origin Authentication
     is the provision of the property that the claimed identity of the
     user on whose behalf received data was originated is corroborated.

   - Data Confidentiality
     is the provision of the property that information is not made
     available or disclosed to unauthorized individuals, entities, or
     processes.

   - Message timeliness and limited replay protection
     is the provision of the property that a message whose generation
     time is outside of a specified time window is not accepted. Note
     that message reordering is not dealt with and can occur in normal
     conditions too.

   For the protocols specified in this memo, it is not possible to
   assure the specific originator of a received SNMP message; rather, it
   is the user on whose behalf the message was originated that is
   authenticated.

   For these protocols, it not possible to obtain data integrity without
   data origin authentication, nor is it possible to obtain data origin
   authentication without data integrity. Further, there is no
   provision for data confidentiality without both data integrity and
   data origin authentication.

   The security protocols used in this memo are considered acceptably
   secure at the time of writing. However, the procedures allow for new
   authentication and privacy methods to be specified at a future time
   if the need arises.

1.4. Module Organization

   The security protocols defined in this memo are split in three
   different modules and each has its specific responsibilities such
   that together they realize the goals and security services described
   above:

   - The authentication module MUST provide for:

     - Data Integrity,

     - Data Origin Authentication

   - The timeliness module MUST provide for:

     - Protection against message delay or replay (to an extent
       greater than can occur through normal operation)

   - The privacy module MUST provide for

     - Protection against disclosure of the message payload.

   The timeliness module is fixed for the User-based Security Model
   while there is provision for multiple authentication and/or privacy
   modules, each of which implements a specific authentication or
   privacy protocol respectively.

1.4.1. Timeliness Module

   Section 3 (Elements of Procedure) uses the timeliness values in an
   SNMP message to do timeliness checking. The timeliness check is only
   performed if authentication is applied to the message. Since the

complete message is checked for integrity, we can assume that the
timeliness values in a message that passes the authentication module
are trustworthy.

1.4.2. Authentication Protocol

Section 6 describes the HMAC-MD5-96 authentication protocol which is
the first authentication protocol that MUST be supported with the
User-based Security Model. Section 7 describes the HMAC-SHA-96
authentication protocol which is another authentication protocol that
SHOULD be supported with the User-based Security Model. In the
future additional or replacement authentication protocols may be
defined as new needs arise.

The User-based Security Model prescribes that, if authentication is
used, then the complete message is checked for integrity in the
authentication module.

For a message to be authenticated, it needs to pass authentication
check by the authentication module and the timeliness check which is
a fixed part of this User-based Security model.

1.4.3. Privacy Protocol

Section 8 describes the CBC-DES Symmetric Encryption Protocol which
is the first privacy protocol to be used with the User-based Security
Model. In the future additional or replacement privacy protocols may
be defined as new needs arise.

The User-based Security Model prescribes that the scopedPDU is
protected from disclosure when a message is sent with privacy.

The User-based Security Model also prescribes that a message needs to
be authenticated if privacy is in use.

1.5. Protection against Message Replay, Delay and Redirection

1.5.1. Authoritative SNMP engine

In order to protect against message replay, delay and redirection,
one of the SNMP engines involved in each communication is designated
to be the authoritative SNMP engine. When an SNMP message contains a
payload which expects a response (those messages that contain a
Confirmed Class PDU [RFC2571]), then the receiver of such messages is
authoritative. When an SNMP message contains a payload which does
not expect a response (those messages that contain an Unconfirmed
Class PDU [RFC2571]), then the sender of such a message is
authoritative.

1.5.2. Mechanisms

   The following mechanisms are used:

   1) To protect against the threat of message delay or replay (to an
      extent greater than can occur through normal operation), a set of
      timeliness indicators (for the authoritative SNMP engine) are
      included in each message generated. An SNMP engine evaluates the
      timeliness indicators to determine if a received message is
      recent. An SNMP engine may evaluate the timeliness indicators to
      ensure that a received message is at least as recent as the last
      message it received from the same source. A non-authoritative
      SNMP engine uses received authentic messages to advance its notion
      of the timeliness indicators at the remote authoritative source.

      An SNMP engine MUST also use a mechanism to match incoming
      Responses to outstanding Requests and it MUST drop any Responses
      that do not match an outstanding request. For example, a msgID can
      be inserted in every message to cater for this functionality.

      These mechanisms provide for the detection of authenticated
      messages whose time of generation was not recent.

      This protection against the threat of message delay or replay does
      not imply nor provide any protection against unauthorized deletion
      or suppression of messages. Also, an SNMP engine may not be able
      to detect message reordering if all the messages involved are sent
      within the Time Window interval. Other mechanisms defined
      independently of the security protocol can also be used to detect
      the re-ordering replay, deletion, or suppression of messages
      containing Set operations (e.g., the MIB variable snmpSetSerialNo
      [RFC1907]).

   2) Verification that a message sent to/from one authoritative SNMP
      engine cannot be replayed to/as-if-from another authoritative SNMP
      engine.

      Included in each message is an identifier unique to the
      authoritative SNMP engine associated with the sender or intended
      recipient of the message.

      A message containing an Unconfirmed Class PDU sent by an
      authoritative SNMP engine to one non-authoritative SNMP engine can
      potentially be replayed to another non-authoritative SNMP engine.
      The latter non-authoritative SNMP engine might (if it knows about
      the same userName with the same secrets at the authoritative SNMP
      engine) as a result update its notion of timeliness indicators of
      the authoritative SNMP engine, but that is not considered a

       threat. In this case, A Report or Response message will be
       discarded by the Message Processing Model, because there should
       not be an outstanding Request message. A Trap will possibly be
       accepted. Again, that is not considered a threat, because the
       communication was authenticated and timely. It is as if the
       authoritative SNMP engine was configured to start sending Traps to
       the second SNMP engine, which theoretically can happen without the
       knowledge of the second SNMP engine anyway. Anyway, the second
       SNMP engine may not expect to receive this Trap, but is allowed to
       see the management information contained in it.

   3) Detection of messages which were not recently generated.

       A set of time indicators are included in the message, indicating
       the time of generation. Messages without recent time indicators
       are not considered authentic. In addition, an SNMP engine MUST
       drop any Responses that do not match an outstanding request. This
       however is the responsibility of the Message Processing Model.

   This memo allows the same user to be defined on multiple SNMP
   engines. Each SNMP engine maintains a value, snmpEngineID, which
   uniquely identifies the SNMP engine. This value is included in each
   message sent to/from the SNMP engine that is authoritative (see
   section 1.5.1). On receipt of a message, an authoritative SNMP
   engine checks the value to ensure that it is the intended recipient,
   and a non-authoritative SNMP engine uses the value to ensure that the
   message is processed using the correct state information.

   Each SNMP engine maintains two values, snmpEngineBoots and
   snmpEngineTime, which taken together provide an indication of time at
   that SNMP engine. Both of these values are included in an
   authenticated message sent to/received from that SNMP engine. On
   receipt, the values are checked to ensure that the indicated
   timeliness value is within a Time Window of the current time. The
   Time Window represents an administrative upper bound on acceptable
   delivery delay for protocol messages.

   For an SNMP engine to generate a message which an authoritative SNMP
   engine will accept as authentic, and to verify that a message
   received from that authoritative SNMP engine is authentic, such an
   SNMP engine must first achieve timeliness synchronization with the
   authoritative SNMP engine. See section 2.3.

1.6. Abstract Service Interfaces

   Abstract service interfaces have been defined to describe the
   conceptual interfaces between the various subsystems within an SNMP
   entity. Similarly a set of abstract service interfaces have been

   defined within the User-based Security Model (USM) to describe the
   conceptual interfaces between the generic USM services and the self-
   contained authentication and privacy services.

   These abstract service interfaces are defined by a set of primitives
   that define the services provided and the abstract data elements that
   must be passed when the services are invoked. This section lists the
   primitives that have been defined for the User-based Security Model.

1.6.1. User-based Security Model Primitives for Authentication

   The User-based Security Model provides the following internal
   primitives to pass data back and forth between the Security Model
   itself and the authentication service:

   statusInformation =
     authenticateOutgoingMsg(
     IN   authKey   -- secret key for authentication
     IN   wholeMsg   -- unauthenticated complete message
     OUT   authenticatedWholeMsg   -- complete authenticated message
)

   statusInformation =
     authenticateIncomingMsg(
     IN   authKey   -- secret key for authentication
     IN   authParameters   -- as received on the wire
     IN   wholeMsg   -- as received on the wire
     OUT   authenticatedWholeMsg   -- complete authenticated message
)

1.6.2. User-based Security Model Primitives for Privacy

   The User-based Security Model provides the following internal
   primitives to pass data back and forth between the Security Model
   itself and the privacy service:

   statusInformation =
     encryptData(
     IN   encryptKey   -- secret key for encryption
     IN   dataToEncrypt   -- data to encrypt (scopedPDU)
     OUT   encryptedData   -- encrypted data (encryptedPDU)
     OUT   privParameters   -- filled in by service provider
)

   statusInformation =
     decryptData(
     IN   decryptKey   -- secret key for decrypting
     IN   privParameters   -- as received on the wire

```
   IN     encryptedData               -- encrypted data (encryptedPDU)
   OUT    decryptedData               -- decrypted data (scopedPDU)
             )
```

2. Elements of the Model

   This section contains definitions required to realize the security
   model defined by this memo.

2.1. User-based Security Model Users

   Management operations using this Security Model make use of a defined
   set of user identities. For any user on whose behalf management
   operations are authorized at a particular SNMP engine, that SNMP
   engine must have knowledge of that user. An SNMP engine that wishes
   to communicate with another SNMP engine must also have knowledge of a
   user known to that engine, including knowledge of the applicable
   attributes of that user.

   A user and its attributes are defined as follows:

   userName
     A string representing the name of the user.

   securityName
     A human-readable string representing the user in a format that is
     Security Model independent.

   authProtocol
     An indication of whether messages sent on behalf of this user can
     be authenticated, and if so, the type of authentication protocol
     which is used. Two such protocols are defined in this memo:

       - the HMAC-MD5-96 authentication protocol.
       - the HMAC-SHA-96 authentication protocol.

   authKey
     If messages sent on behalf of this user can be authenticated,
     the (private) authentication key for use with the authentication
     protocol. Note that a user's authentication key will normally
     be different at different authoritative SNMP engines. The authKey
     is not accessible via SNMP. The length requirements of the authKey
     are defined by the authProtocol in use.

   authKeyChange and authOwnKeyChange
     The only way to remotely update the authentication key. Does
     that in a secure manner, so that the update can be completed
     without the need to employ privacy protection.

   privProtocol
      An indication of whether messages sent on behalf of this user
      can be protected from disclosure, and if so, the type of privacy
      protocol which is used. One such protocol is defined in this
      memo: the CBC-DES Symmetric Encryption Protocol.

   privKey
      If messages sent on behalf of this user can be en/decrypted,
      the (private) privacy key for use with the privacy protocol.
      Note that a user's privacy key will normally be different at
      different authoritative SNMP engines. The privKey is not
      accessible via SNMP. The length requirements of the privKey are
      defined by the privProtocol in use.

   privKeyChange and privOwnKeyChange
      The only way to remotely update the encryption key. Does that
      in a secure manner, so that the update can be completed without
      the need to employ privacy protection.

2.2. Replay Protection

   Each SNMP engine maintains three objects:

   - snmpEngineID, which (at least within an administrative domain)
     uniquely and unambiguously identifies an SNMP engine.

   - snmpEngineBoots, which is a count of the number of times the
     SNMP engine has re-booted/re-initialized since snmpEngineID
     was last configured; and,

   - snmpEngineTime, which is the number of seconds since the
     snmpEngineBoots counter was last incremented.

   Each SNMP engine is always authoritative with respect to these
   objects in its own SNMP entity. It is the responsibility of a
   non-authoritative SNMP engine to synchronize with the
   authoritative SNMP engine, as appropriate.

   An authoritative SNMP engine is required to maintain the values of
   its snmpEngineID and snmpEngineBoots in non-volatile storage.

2.2.1. msgAuthoritativeEngineID

   The msgAuthoritativeEngineID value contained in an authenticated
   message is used to defeat attacks in which messages from one SNMP
   engine to another SNMP engine are replayed to a different SNMP
   engine. It represents the snmpEngineID at the authoritative SNMP
   engine involved in the exchange of the message.

When an authoritative SNMP engine is first installed, it sets its
local value of snmpEngineID according to a enterprise-specific
algorithm (see the definition of the Textual Convention for
SnmpEngineID in the SNMP Architecture document [RFC2571]).

2.2.2. msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime

The msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime
values contained in an authenticated message are used to defeat
attacks in which messages are replayed when they are no longer
valid. They represent the snmpEngineBoots and snmpEngineTime
values at the authoritative SNMP engine involved in the exchange
of the message.

Through use of snmpEngineBoots and snmpEngineTime, there is no
requirement for an SNMP engine to have a non-volatile clock which
ticks (i.e., increases with the passage of time) even when the
SNMP engine is powered off. Rather, each time an SNMP engine
re-boots, it retrieves, increments, and then stores snmpEngineBoots
in non-volatile storage, and resets snmpEngineTime to zero.

When an SNMP engine is first installed, it sets its local values
of snmpEngineBoots and snmpEngineTime to zero. If snmpEngineTime
ever reaches its maximum value (2147483647), then snmpEngineBoots
is incremented as if the SNMP engine has re-booted and
snmpEngineTime is reset to zero and starts incrementing again.

Each time an authoritative SNMP engine re-boots, any SNMP engines
holding that authoritative SNMP engine's values of snmpEngineBoots
and snmpEngineTime need to re-synchronize prior to sending
correctly authenticated messages to that authoritative SNMP engine
(see Section 2.3 for (re-)synchronization procedures). Note,
however, that the procedures do provide for a notification to be
accepted as authentic by a receiving SNMP engine, when sent by an
authoritative SNMP engine which has re-booted since the receiving
SNMP engine last (re-)synchronized.

If an authoritative SNMP engine is ever unable to determine its
latest snmpEngineBoots value, then it must set its snmpEngineBoots
value to 2147483647.

Whenever the local value of snmpEngineBoots has the value 2147483647
it latches at that value and an authenticated message always causes
an notInTimeWindow authentication failure.

In order to reset an SNMP engine whose snmpEngineBoots value has
reached the value 2147483647, manual intervention is required.
The engine must be physically visited and re-configured, either

with a new snmpEngineID value, or with new secret values for the
authentication and privacy protocols of all users known to that
SNMP engine. Note that even if an SNMP engine re-boots once a second
that it would still take approximately 68 years before the max value
of 2147483647 would be reached.

2.2.3. Time Window

The Time Window is a value that specifies the window of time in
which a message generated on behalf of any user is valid. This
memo specifies that the same value of the Time Window, 150 seconds,
is used for all users.

2.3. Time Synchronization

Time synchronization, required by a non-authoritative SNMP engine
in order to proceed with authentic communications, has occurred
when the non-authoritative SNMP engine has obtained a local notion
of the authoritative SNMP engine's values of snmpEngineBoots and
snmpEngineTime from the authoritative SNMP engine. These values
must be (and remain) within the authoritative SNMP engine's Time
Window. So the local notion of the authoritative SNMP engine's
values must be kept loosely synchronized with the values stored
at the authoritative SNMP engine. In addition to keeping a local
copy of snmpEngineBoots and snmpEngineTime from the authoritative
SNMP engine, a non-authoritative SNMP engine must also keep one
local variable, latestReceivedEngineTime. This value records the
highest value of snmpEngineTime that was received by the
non-authoritative SNMP engine from the authoritative SNMP engine
and is used to eliminate the possibility of replaying messages
that would prevent the non-authoritative SNMP engine's notion of
the snmpEngineTime from advancing.

A non-authoritative SNMP engine must keep local notions of these
values
(snmpEngineBoots, snmpEngineTime and latestReceivedEngineTime)
for each authoritative SNMP engine with which it wishes to
communicate. Since each authoritative SNMP engine is uniquely
and unambiguously identified by its value of snmpEngineID, the
non-authoritative SNMP engine may use this value as a key in
order to cache its local notions of these values.

Time synchronization occurs as part of the procedures of receiving
an SNMP message (Section 3.2, step 7b). As such, no explicit time
synchronization procedure is required by a non-authoritative SNMP
engine. Note, that whenever the local value of snmpEngineID is
changed (e.g., through discovery) or when secure communications
are first established with an authoritative SNMP engine, the local

   values of snmpEngineBoots and latestReceivedEngineTime should be
   set to zero. This will cause the time synchronization to occur
   when the next authentic message is received.

2.4. SNMP Messages Using this Security Model

   The syntax of an SNMP message using this Security Model adheres
   to the message format defined in the version-specific Message
   Processing Model document (for example [RFC2572]).

   The field msgSecurityParameters in SNMPv3 messages has a data type
   of OCTET STRING. Its value is the BER serialization of the
   following ASN.1 sequence:

   USMSecurityParametersSyntax DEFINITIONS IMPLICIT TAGS ::= BEGIN

      UsmSecurityParameters ::=
          SEQUENCE {
            -- global User-based security parameters
              msgAuthoritativeEngineID   OCTET STRING,
              msgAuthoritativeEngineBoots INTEGER (0..2147483647),
              msgAuthoritativeEngineTime   INTEGER (0..2147483647),
              msgUserName   OCTET STRING (SIZE(0..32)),
            -- authentication protocol specific parameters
              msgAuthenticationParameters OCTET STRING,
            -- privacy protocol specific parameters
              msgPrivacyParameters   OCTET STRING
          }
      END

   The fields of this sequence are:

   - The msgAuthoritativeEngineID specifies the snmpEngineID of the
     authoritative SNMP engine involved in the exchange of the message.

   - The msgAuthoritativeEngineBoots specifies the snmpEngineBoots value
     at the authoritative SNMP engine involved in the exchange of the
     message.

   - The msgAuthoritativeEngineTime specifies the snmpEngineTime value
     at the authoritative SNMP engine involved in the exchange of the
     message.

   - The msgUserName specifies the user (principal) on whose behalf the
     message is being exchanged. Note that a zero-length userName will
     not match any user, but it can be used for snmpEngineID discovery.

   - The msgAuthenticationParameters are defined by the authentication
     protocol in use for the message, as defined by the
     usmUserAuthProtocol column in the user's entry in the usmUserTable.

   - The msgPrivacyParameters are defined by the privacy protocol in use
     for the message, as defined by the usmUserPrivProtocol column in
     the user's entry in the usmUserTable).

   See appendix A.4 for an example of the BER encoding of field
   msgSecurityParameters.

2.5. Services provided by the User-based Security Model

   This section describes the services provided by the User-based
   Security Model with their inputs and outputs.

   The services are described as primitives of an abstract service
   interface and the inputs and outputs are described as abstract data
   elements as they are passed in these abstract service primitives.

2.5.1. Services for Generating an Outgoing SNMP Message

   When the Message Processing (MP) Subsystem invokes the User-based
   Security module to secure an outgoing SNMP message, it must use the
   appropriate service as provided by the Security module. These two
   services are provided:

   1) A service to generate a Request message. The abstract service
      primitive is:

      statusInformation =   -- success or errorIndication
        generateRequestMsg(
        IN   messageProcessingModel -- typically, SNMP version
        IN   globalData   -- message header, admin data
        IN   maxMessageSize   -- of the sending SNMP entity
        IN   securityModel   -- for the outgoing message
        IN   securityEngineID   -- authoritative SNMP entity
        IN   securityName   -- on behalf of this principal
        IN   securityLevel   -- Level of Security requested
        IN   scopedPDU   -- message (plaintext) payload
        OUT   securityParameters   -- filled in by Security Module
        OUT   wholeMsg   -- complete generated message
        OUT   wholeMsgLength   -- length of generated message
)

   2) A service to generate a Response message. The abstract service
      primitive is:

```
      statusInformation =              -- success or errorIndication
        generateResponseMsg(
        IN   messageProcessingModel   -- typically, SNMP version
        IN   globalData               -- message header, admin data
        IN   maxMessageSize           -- of the sending SNMP entity
        IN   securityModel            -- for the outgoing message
        IN   securityEngineID         -- authoritative SNMP entity
        IN   securityName             -- on behalf of this principal
        IN   securityLevel            -- Level of Security requested
        IN   scopedPDU                -- message (plaintext) payload
        IN   securityStateReference   -- reference to security state
                                      -- information from original
                                      -- request
        OUT  securityParameters       -- filled in by Security Module
        OUT  wholeMsg                 -- complete generated message
        OUT  wholeMsgLength           -- length of generated message
             )
```

   The abstract data elements passed as parameters in the abstract
   service primitives are as follows:

    statusInformation
      An indication of whether the encoding and securing of the message
      was successful. If not it is an indication of the problem.
    messageProcessingModel
      The SNMP version number for the message to be generated. This
      data is not used by the User-based Security module.
    globalData
      The message header (i.e., its administrative information). This
      data is not used by the User-based Security module.
    maxMessageSize
      The maximum message size as included in the message. This data is
      not used by the User-based Security module.
    securityParameters
      These are the security parameters. They will be filled in by the
      User-based Security module.

    securityModel
      The securityModel in use. Should be User-based Security Model.
      This data is not used by the User-based Security module.
    securityName
      Together with the snmpEngineID it identifies a row in the
      usmUserTable that is to be used for securing the message. The
      securityName has a format that is independent of the Security
      Model. In case of a response this parameter is ignored and the
      value from the cache is used.

```

       securityLevel
         The Level of Security from which the User-based Security module
         determines if the message needs to be protected from disclosure
         and if the message needs to be authenticated.
       securityEngineID
         The snmpEngineID of the authoritative SNMP engine to which a
         Request message is to be sent. In case of a response it is implied
         to be the processing SNMP engine's snmpEngineID and so if it is
         specified, then it is ignored.
       scopedPDU
         The message payload.  The data is opaque as far as the User-based
         Security Model is concerned.
       securityStateReference
         A handle/reference to cachedSecurityData to be used when securing
         an outgoing Response message.  This is the exact same
         handle/reference as it was generated by the User-based Security
         module when processing the incoming Request message to which this
         is the Response message.
       wholeMsg
         The fully encoded and secured message ready for sending on the
         wire.
       wholeMsgLength
         The length of the encoded and secured message (wholeMsg).

   Upon completion of the process, the User-based Security module
   returns statusInformation. If the process was successful, the
   completed message with privacy and authentication applied if such was
   requested by the specified securityLevel is returned. If the process
   was not successful, then an errorIndication is returned.

2.5.2.  Services for Processing an Incoming SNMP Message

   When the Message Processing (MP) Subsystem invokes the User-based
   Security module to verify proper security of an incoming message, it
   must use the service provided for an incoming message. The abstract
   service primitive is:

   statusInformation =                  -- errorIndication or success
                                        -- error counter OID/value if error
     processIncomingMsg(
     IN    messageProcessingModel   -- typically, SNMP version
     IN    maxMessageSize           -- of the sending SNMP entity
     IN    securityParameters       -- for the received message
     IN    securityModel            -- for the received message
     IN    securityLevel            -- Level of Security
     IN    wholeMsg                 -- as received on the wire
     IN    wholeMsgLength           -- length as received on the wire
     OUT   securityEngineID         -- authoritative SNMP entity

```
     OUT   securityName   -- identification of the principal
     OUT   scopedPDU,   -- message (plaintext) payload
     OUT   maxSizeResponseScopedPDU -- maximum size of the Response PDU
     OUT   securityStateReference   -- reference to security state
)   -- information, needed for response
```

   The abstract data elements passed as parameters in the abstract
   service primitives are as follows:

    statusInformation
      An indication of whether the process was successful or not.  If
      not, then the statusInformation includes the OID and the value of
      the error counter that was incremented.
    messageProcessingModel
      The SNMP version number as received in the message.  This data is
      not used by the User-based Security module.
    maxMessageSize
      The maximum message size as included in the message.  The User-
      based Security module uses this value to calculate the
      maxSizeResponseScopedPDU.
    securityParameters
      These are the security parameters as received in the message.
    securityModel
      The securityModel in use.  Should be the User-based Security
      Model.  This data is not used by the User-based Security module.
    securityLevel
      The Level of Security from which the User-based Security module
      determines if the message needs to be protected from disclosure
      and if the message needs to be authenticated.
    wholeMsg
      The whole message as it was received.
    wholeMsgLength
      The length of the message as it was received (wholeMsg).
    securityEngineID
      The snmpEngineID that was extracted from the field
      msgAuthoritativeEngineID and that was used to lookup the secrets
      in the usmUserTable.
    securityName
      The security name representing the user on whose behalf the
      message was received.  The securityName has a format that is
      independent of the Security Model.
    scopedPDU
      The message payload.  The data is opaque as far as the User-based
      Security Model is concerned.

   maxSizeResponseScopedPDU
     The maximum size of a scopedPDU to be included in a possible
     Response message.  The User-based Security module calculates this
     size based on the msgMaxSize (as received in the message) and the
     space required for the message header (including the
     securityParameters) for such a Response message.
   securityStateReference
     A handle/reference to cachedSecurityData to be used when securing
     an outgoing Response message.  When the Message Processing
     Subsystem calls the User-based Security module to generate a
     response to this incoming message it must pass this
     handle/reference.

   Upon completion of the process, the User-based Security module
   returns statusInformation and, if the process was successful, the
   additional data elements for further processing of the message.  If
   the process was not successful, then an errorIndication, possibly
   with a OID and value pair of an error counter that was incremented.

2.6.  Key Localization Algorithm.

   A localized key is a secret key shared between a user U and one
   authoritative SNMP engine E.  Even though a user may have only one
   password and therefore one key for the whole network, the actual
   secrets shared between the user and each authoritative SNMP engine
   will be different. This is achieved by key localization [Localized-
   key].

   First, if a user uses a password, then the user's password is
   converted into a key Ku using one of the two algorithms described in
   Appendices A.2.1 and A.2.2.

   To convert key Ku into a localized key Kul of user U at the
   authoritative SNMP engine E, one appends the snmpEngineID of the
   authoritative SNMP engine to the key Ku and then appends the key Ku
   to the result, thus enveloping the snmpEngineID within the two copies
   of user's key Ku. Then one runs a secure hash function (which one
   depends on the authentication protocol defined for this user U at
   authoritative SNMP engine E; this document defines two authentication
   protocols with their associated algorithms based on MD5 and SHA). The
   output of the hash-function is the localized key Kul for user U at
   the authoritative SNMP engine E.

3.  Elements of Procedure

   This section describes the security related procedures followed by an
   SNMP engine when processing SNMP messages according to the User-based
   Security Model.

3.1.  Generating an Outgoing SNMP Message

   This section describes the procedure followed by an SNMP engine
   whenever it generates a message containing a management operation
   (like a request, a response, a notification, or a report) on behalf
   of a user, with a particular securityLevel.

   1)  a) If any securityStateReference is passed (Response or Report
          message), then information concerning the user is extracted
          from the cachedSecurityData.  The cachedSecurityData can now
          be discarded.  The securityEngineID is set to the local
          snmpEngineID.  The securityLevel is set to the value specified
          by the calling module.

          Otherwise,

       b) based on the securityName, information concerning the user at
          the destination snmpEngineID, specified by the
          securityEngineID, is extracted from the Local Configuration
          Datastore (LCD, usmUserTable). If information about the user
          is absent from the LCD, then an error indication
          (unknownSecurityName) is returned to the calling module.

   2)  If the securityLevel specifies that the message is to be
       protected from disclosure, but the user does not support both an
       authentication and a privacy protocol then the message cannot be
       sent.  An error indication (unsupportedSecurityLevel) is returned
       to the calling module.

   3)  If the securityLevel specifies that the message is to be
       authenticated, but the user does not support an authentication
       protocol, then the message cannot be sent. An error indication
       (unsupportedSecurityLevel) is returned to the calling module.

   4)  a) If the securityLevel specifies that the message is to be
          protected from disclosure, then the octet sequence
          representing the serialized scopedPDU is encrypted according
          to the user's privacy protocol. To do so a call is made to the
          privacy module that implements the user's privacy protocol
          according to the abstract primitive:

          statusInformation =        -- success or failure
            encryptData(
            IN     encryptKey         -- user's localized privKey
            IN     dataToEncrypt      -- serialized scopedPDU
            OUT    encryptedData      -- serialized encryptedPDU
            OUT    privParameters     -- serialized privacy parameters
                   )

statusInformation
  indicates if the encryption process was successful or not.
encryptKey
  the user's localized private privKey is the secret key that
  can be used by the encryption algorithm.
dataToEncrypt
  the serialized scopedPDU is the data to be encrypted.
encryptedData
  the encryptedPDU represents the encrypted scopedPDU,
  encoded as an OCTET STRING.
privParameters
  the privacy parameters, encoded as an OCTET STRING.

If the privacy module returns failure, then the message cannot
be sent and an error indication (encryptionError) is returned
to the calling module.

If the privacy module returns success, then the returned
privParameters are put into the msgPrivacyParameters field of
the securityParameters and the encryptedPDU serves as the
payload of the message being prepared.

Otherwise,

b) If the securityLevel specifies that the message is not to be
   be protected from disclosure, then a zero-length OCTET STRING
   is encoded into the msgPrivacyParameters field of the
   securityParameters and the plaintext scopedPDU serves as the
   payload of the message being prepared.

5)  The securityEngineID is encoded as an OCTET STRING into the
    msgAuthoritativeEngineID field of the securityParameters.  Note
    that an empty (zero length) securityEngineID is OK for a Request
    message, because that will cause the remote (authoritative) SNMP
    engine to return a Report PDU with the proper securityEngineID
    included in the msgAuthoritativeEngineID in the
    securityParameters of that returned Report PDU.

6)  a) If the securityLevel specifies that the message is to be
       authenticated, then the current values of snmpEngineBoots and
       snmpEngineTime corresponding to the securityEngineID from the
       LCD are used.

       Otherwise,

    b) If this is a Response or Report message, then the current
       value of snmpEngineBoots and snmpEngineTime corresponding to
       the local snmpEngineID from the LCD are used.

Otherwise,

c) If this is a Request message, then a zero value is used for
   both snmpEngineBoots and snmpEngineTime. This zero value gets
   used if snmpEngineID is empty.

The values are encoded as INTEGER respectively into the
msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime fields
of the securityParameters.

7) The userName is encoded as an OCTET STRING into the msgUserName
   field of the securityParameters.

8) a) If the securityLevel specifies that the message is to be
      authenticated, the message is authenticated according to the
      user's authentication protocol. To do so a call is made to the
      authentication module that implements the user's
      authentication protocol according to the abstract service
      primitive:

      ```
      statusInformation =
        authenticateOutgoingMsg(
        IN  authKey                -- the user's localized authKey
        IN  wholeMsg               -- unauthenticated message
        OUT authenticatedWholeMsg -- authenticated complete message
             )
      ```

      statusInformation
        indicates if authentication was successful or not.
      authKey
        the user's localized private authKey is the secret key that
        can be used by the authentication algorithm.
      wholeMsg
        the complete serialized message to be authenticated.
      authenticatedWholeMsg
        the same as the input given to the authenticateOutgoingMsg
        service, but with msgAuthenticationParameters properly
        filled in.

      If the authentication module returns failure, then the message
      cannot be sent and an error indication (authenticationFailure)
      is returned to the calling module.

      If the authentication module returns success, then the
      msgAuthenticationParameters field is put into the
      securityParameters and the authenticatedWholeMsg represents
      the serialization of the authenticated message being prepared.

         Otherwise,

     b) If the securityLevel specifies that the message is not to be
        authenticated then a zero-length OCTET STRING is encoded into
        the msgAuthenticationParameters field of the
        securityParameters. The wholeMsg is now serialized and then
        represents the unauthenticated message being prepared.

  9) The completed message with its length is returned to the calling
      module with the statusInformation set to success.

3.2. Processing an Incoming SNMP Message

   This section describes the procedure followed by an SNMP engine
   whenever it receives a message containing a management operation on
   behalf of a user, with a particular securityLevel.

   To simplify the elements of procedure, the release of state
   information is not always explicitly specified. As a general rule, if
   state information is available when a message gets discarded, the
   state information should also be released. Also, an error indication
   can return an OID and value for an incremented counter and optionally
   a value for securityLevel, and values for contextEngineID or
   contextName for the counter. In addition, the securityStateReference
   data is returned if any such information is available at the point
   where the error is detected.

  1) If the received securityParameters is not the serialization
      (according to the conventions of [RFC1906]) of an OCTET STRING
      formatted according to the UsmSecurityParameters defined in
      section 2.4, then the snmpInASNParseErrs counter [RFC1907] is
      incremented, and an error indication (parseError) is returned to
      the calling module. Note that we return without the OID and
      value of the incremented counter, because in this case there is
      not enough information to generate a Report PDU.

  2) The values of the security parameter fields are extracted from
      the securityParameters. The securityEngineID to be returned to
      the caller is the value of the msgAuthoritativeEngineID field.
      The cachedSecurityData is prepared and a securityStateReference
      is prepared to reference this data. Values to be cached are:

          msgUserName

  3) If the value of the msgAuthoritativeEngineID field in the
      securityParameters is unknown then:

      a) a non-authoritative SNMP engine that performs discovery may
         optionally create a new entry in its Local Configuration
         Datastore (LCD) and continue processing;

         or

      b) the usmStatsUnknownEngineIDs counter is incremented, and
         an error indication (unknownEngineID) together with the
         OID and value of the incremented counter is returned to
         the calling module.

     Note in the event that a zero-length, or other illegally
     sized msgAuthoritativeEngineID is received, b) should be
     chosen to facilitate engineID discovery.
     Otherwise the choice between a) and b) is an implementation
     issue.

  4) Information about the value of the msgUserName and
      msgAuthoritativeEngineID fields is extracted from the Local
      Configuration Datastore (LCD, usmUserTable). If no information
      is available for the user, then the usmStatsUnknownUserNames
      counter is incremented and an error indication
      (unknownSecurityName) together with the OID and value of the
      incremented counter is returned to the calling module.

  5) If the information about the user indicates that it does not
      support the securityLevel requested by the caller, then the
      usmStatsUnsupportedSecLevels counter is incremented and an
      error indication (unsupportedSecurityLevel) together with the
      OID and value of the incremented counter is returned to the
      calling module.

  6) If the securityLevel specifies that the message is to be
      authenticated, then the message is authenticated according to
      the user's authentication protocol. To do so a call is made
      to the authentication module that implements the user's
      authentication protocol according to the abstract service
      primitive:

```
statusInformation =          -- success or failure
  authenticateIncomingMsg(
  IN    authKey              -- the user's localized authKey
  IN    authParameters       -- as received on the wire
  IN    wholeMsg             -- as received on the wire
  OUT   authenticatedWholeMsg -- checked for authentication
          )
```

```
         statusInformation
           indicates if authentication was successful or not.
         authKey
           the user's localized private authKey is the secret key that
           can be used by the authentication algorithm.
         wholeMsg
           the complete serialized message to be authenticated.
         authenticatedWholeMsg
           the same as the input given to the authenticateIncomingMsg
           service, but after authentication has been checked.
```

         If the authentication module returns failure, then the message
         cannot be trusted, so the usmStatsWrongDigests counter is
         incremented and an error indication (authenticationFailure)
         together with the OID and value of the incremented counter is
         returned to the calling module.

         If the authentication module returns success, then the message
         is authentic and can be trusted so processing continues.

     7) If the securityLevel indicates an authenticated message, then
         the local values of snmpEngineBoots, snmpEngineTime
         and latestReceivedEngineTime
         corresponding to the value of the msgAuthoritativeEngineID
         field are extracted from the Local Configuration Datastore.

         a) If the extracted value of msgAuthoritativeEngineID is the
            same as the value of snmpEngineID of the processing SNMP
            engine (meaning this is the authoritative SNMP engine),
            then if any of the following conditions is true, then the
            message is considered to be outside of the Time Window:

              - the local value of snmpEngineBoots is 2147483647;

              - the value of the msgAuthoritativeEngineBoots field differs
                from the local value of snmpEngineBoots; or,

              - the value of the msgAuthoritativeEngineTime field differs
                from the local notion of snmpEngineTime by more than
                +/- 150 seconds.

            If the message is considered to be outside of the Time Window
            then the usmStatsNotInTimeWindows counter is incremented and
            an error indication (notInTimeWindow) together with the OID,
            the value of the incremented counter, and an indication that
            the error must be reported with a securityLevel of authNoPriv,
            is returned to the calling module

      b) If the extracted value of msgAuthoritativeEngineID is not the
         same as the value snmpEngineID of the processing SNMP engine
         (meaning this is not the authoritative SNMP engine), then:

         1) if at least one of the following conditions is true:

            - the extracted value of the msgAuthoritativeEngineBoots
              field is greater than the local notion of the value of
              snmpEngineBoots; or,

            - the extracted value of the msgAuthoritativeEngineBoots
              field is equal to the local notion of the value of
              snmpEngineBoots, and the extracted value of
              msgAuthoritativeEngineTime field is greater than the
              value of latestReceivedEngineTime,

            then the LCD entry corresponding to the extracted value
            of the msgAuthoritativeEngineID field is updated, by
            setting:

               - the local notion of the value of snmpEngineBoots to
                 the value of the msgAuthoritativeEngineBoots field,
               - the local notion of the value of snmpEngineTime to
                 the value of the msgAuthoritativeEngineTime field,
                 and
               - the latestReceivedEngineTime to the value of the
                 value of the msgAuthoritativeEngineTime field.

         2) if any of the following conditions is true, then the
            message is considered to be outside of the Time Window:

            - the local notion of the value of snmpEngineBoots is
              2147483647;

            - the value of the msgAuthoritativeEngineBoots field is
              less than the local notion of the value of
              snmpEngineBoots; or,

            - the value of the msgAuthoritativeEngineBoots field is
              equal to the local notion of the value of
              snmpEngineBoots and the value of the
              msgAuthoritativeEngineTime field is more than 150
              seconds less than the local notion of the value of
              snmpEngineTime.

            If the message is considered to be outside of the Time
            Window then an error indication (notInTimeWindow) is
            returned to the calling module.

        Note that this means that a too old (possibly replayed)
        message has been detected and is deemed unauthentic.

        Note that this procedure allows for the value of
        msgAuthoritativeEngineBoots in the message to be greater
        than the local notion of the value of snmpEngineBoots to
        allow for received messages to be accepted as authentic
        when received from an authoritative SNMP engine that has
        re-booted since the receiving SNMP engine last
        (re-)synchronized.

  8)   a) If the securityLevel indicates that the message was protected
        from disclosure, then the OCTET STRING representing the
        encryptedPDU is decrypted according to the user's privacy
        protocol to obtain an unencrypted serialized scopedPDU value.
        To do so a call is made to the privacy module that implements
        the user's privacy protocol according to the abstract
        primitive:

        statusInformation =   -- success or failure
          decryptData(
          IN   decryptKey   -- the user's localized privKey
          IN   privParameters   -- as received on the wire
          IN   encryptedData   -- encryptedPDU as received
          OUT   decryptedData   -- serialized decrypted scopedPDU
)

        statusInformation
          indicates if the decryption process was successful or not.
        decryptKey
          the user's localized private privKey is the secret key that
          can be used by the decryption algorithm.
        privParameters
          the msgPrivacyParameters, encoded as an OCTET STRING.
        encryptedData
          the encryptedPDU represents the encrypted scopedPDU, encoded
          as an OCTET STRING.
        decryptedData
          the serialized scopedPDU if decryption is successful.

        If the privacy module returns failure, then the message can
        not be processed, so the usmStatsDecryptionErrors counter is
        incremented and an error indication (decryptionError) together
        with the OID and value of the incremented counter is returned
        to the calling module.

If the privacy module returns success, then the decrypted
scopedPDU is the message payload to be returned to the calling
module.

Otherwise,

b) The scopedPDU component is assumed to be in plain text
   and is the message payload to be returned to the calling
   module.

9) The maxSizeResponseScopedPDU is calculated. This is the
    maximum size allowed for a scopedPDU for a possible Response
    message. Provision is made for a message header that allows the
    same securityLevel as the received Request.

10) The securityName for the user is retrieved from the
    usmUserTable.

11) The security data is cached as cachedSecurityData, so that a
    possible response to this message can and will use the same
    authentication and privacy secrets. Information to be
    saved/cached is as follows:

        msgUserName,
        usmUserAuthProtocol, usmUserAuthKey
        usmUserPrivProtocol, usmUserPrivKey

12) The statusInformation is set to success and a return is made to
    the calling module passing back the OUT parameters as specified
    in the processIncomingMsg primitive.

4. Discovery

The User-based Security Model requires that a discovery process
obtains sufficient information about other SNMP engines in order to
communicate with them. Discovery requires an non-authoritative SNMP
engine to learn the authoritative SNMP engine's snmpEngineID value
before communication may proceed. This may be accomplished by
generating a Request message with a securityLevel of noAuthNoPriv, a
msgUserName of zero-length, a msgAuthoritativeEngineID value of zero
length, and the varBindList left empty. The response to this message
will be a Report message containing the snmpEngineID of the
authoritative SNMP engine as the value of the
msgAuthoritativeEngineID field within the msgSecurityParameters
field. It contains a Report PDU with the usmStatsUnknownEngineIDs
counter in the varBindList.

   If authenticated communication is required, then the discovery
   process should also establish time synchronization with the
   authoritative SNMP engine. This may be accomplished by sending an
   authenticated Request message with the value of
   msgAuthoritativeEngineID set to the newly learned snmpEngineID and
   with the values of msgAuthoritativeEngineBoots and
   msgAuthoritativeEngineTime set to zero. For an authenticated Request
   message, a valid userName must be used in the msgUserName field. The
   response to this authenticated message will be a Report message
   containing the up to date values of the authoritative SNMP engine's
   snmpEngineBoots and snmpEngineTime as the value of the
   msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime fields
   respectively. It also contains the usmStatsNotInTimeWindows counter
   in the varBindList of the Report PDU. The time synchronization then
   happens automatically as part of the procedures in section 3.2 step
   7b. See also section 2.3.

5. Definitions

SNMP-USER-BASED-SM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    OBJECT-IDENTITY,
    snmpModules, Counter32   FROM SNMPv2-SMI
    TEXTUAL-CONVENTION, TestAndIncr,
    RowStatus, RowPointer,
    StorageType, AutonomousType   FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP   FROM SNMPv2-CONF
    SnmpAdminString, SnmpEngineID,
    snmpAuthProtocols, snmpPrivProtocols FROM SNMP-FRAMEWORK-MIB;

snmpUsmMIB MODULE-IDENTITY
    LAST-UPDATED "9901200000Z"   -- 20 Jan 1999, midnight
    ORGANIZATION "SNMPv3 Working Group"
    CONTACT-INFO "WG-email: snmpv3@lists.tislabs.com
                  Subscribe: majordomo@lists.tislabs.com
                         In msg body: subscribe snmpv3

                  Chair: Russ Mundy
                              Trusted Information Systems
                  postal: 3060 Washington Rd
                              Glenwood MD 21738
                              USA
                  email: mundy@tislabs.com
                  phone: +1-301-854-6889

                  Co-editor   Uri Blumenthal

```
                          IBM T. J. Watson Research
                 postal:    30 Saw Mill River Pkwy,
                            Hawthorne, NY 10532
                            USA
                 email:     uri@watson.ibm.com
                 phone:     +1-914-784-7964

                 Co-editor: Bert Wijnen
                            IBM T. J. Watson Research
                 postal:    Schagen 33
                            3461 GL Linschoten
                            Netherlands
                 email:     wijnen@vnet.ibm.com
                 phone:     +31-348-432-794
                 "
    DESCRIPTION  "The management information definitions for the
                  SNMP User-based Security Model.
                 "
-- Revision history

    REVISION     "9901200000Z"           -- 20 Jan 1999, midnight
    DESCRIPTION  "Clarifications, published as RFC2574"

    REVISION     "9711200000Z"           -- 20 Nov 1997, midnight
    DESCRIPTION  "Initial version, published as RFC2274"

    ::= { snmpModules 15 }

-- Administrative assignments ***************************************

usmMIBObjects     OBJECT IDENTIFIER ::= { snmpUsmMIB 1 }
usmMIBConformance OBJECT IDENTIFIER ::= { snmpUsmMIB 2 }

-- Identification of Authentication and Privacy Protocols ***********

usmNoAuthProtocol OBJECT-IDENTITY
    STATUS       current
    DESCRIPTION  "No Authentication Protocol."
    ::= { snmpAuthProtocols 1 }

usmHMACMD5AuthProtocol OBJECT-IDENTITY
    STATUS       current
    DESCRIPTION  "The HMAC-MD5-96 Digest Authentication Protocol."
    REFERENCE    "- H. Krawczyk, M. Bellare, R. Canetti HMAC:
                    Keyed-Hashing for Message Authentication,
                    RFC2104, Feb 1997.
                  - Rivest, R., Message Digest Algorithm MD5, RFC1321.
                 "
```

```
    ::= { snmpAuthProtocols 2 }


usmHMACSHAAuthProtocol OBJECT-IDENTITY
    STATUS          current
    DESCRIPTION   "The HMAC-SHA-96 Digest Authentication Protocol."
    REFERENCE     "- H. Krawczyk, M. Bellare, R. Canetti, HMAC:
                     Keyed-Hashing for Message Authentication,
                     RFC2104, Feb 1997.
                   - Secure Hash Algorithm. NIST FIPS 180-1.
                  "
    ::= { snmpAuthProtocols 3 }


usmNoPrivProtocol OBJECT-IDENTITY
    STATUS          current
    DESCRIPTION   "No Privacy Protocol."
    ::= { snmpPrivProtocols 1 }


usmDESPrivProtocol OBJECT-IDENTITY
    STATUS          current
    DESCRIPTION   "The CBC-DES Symmetric Encryption Protocol."
    REFERENCE     "- Data Encryption Standard, National Institute of
                     Standards and Technology.  Federal Information
                     Processing Standard (FIPS) Publication 46-1.
                     Supersedes FIPS Publication 46,
                     (January, 1977; reaffirmed January, 1988).

                   - Data Encryption Algorithm, American National
                     Standards Institute.  ANSI X3.92-1981,
                     (December, 1980).

                   - DES Modes of Operation, National Institute of
                     Standards and Technology.  Federal Information
                     Processing Standard (FIPS) Publication 81,
                     (December, 1980).

                   - Data Encryption Algorithm - Modes of Operation,
                     American National Standards Institute.
                     ANSI X3.106-1983, (May 1983).
                  "
    ::= { snmpPrivProtocols 2 }


-- Textual Conventions *********************************************


KeyChange ::=     TEXTUAL-CONVENTION
   STATUS          current
   DESCRIPTION
```

                "Every definition of an object with this syntax must identify
                 a protocol P, a secret key K, and a hash algorithm H
                 that produces output of L octets.

                 The object's value is a manager-generated, partially-random
                 value which, when modified, causes the value of the secret
                 key K, to be modified via a one-way function.

                 The value of an instance of this object is the concatenation
                 of two components: first a 'random' component and then a
                 'delta' component.

                 The lengths of the random and delta components
                 are given by the corresponding value of the protocol P;
                 if P requires K to be a fixed length, the length of both the
                 random and delta components is that fixed length; if P
                 allows the length of K to be variable up to a particular
                 maximum length, the length of the random component is that
                 maximum length and the length of the delta component is any
                 length less than or equal to that maximum length.
                 For example, usmHMACMD5AuthProtocol requires K to be a fixed
                 length of 16 octets and L – of 16 octets.
                 usmHMACSHAAuthProtocol requires K to be a fixed length of
                 20 octets and L – of 20 octets. Other protocols may define
                 other sizes, as deemed appropriate.

                 When a requester wants to change the old key K to a new
                 key keyNew on a remote entity, the 'random' component is
                 obtained from either a true random generator, or from a
                 pseudorandom generator, and the 'delta' component is
                 computed as follows:

                  - a temporary variable is initialized to the existing value
                    of K;
                  - if the length of the keyNew is greater than L octets,
                    then:
                      - the random component is appended to the value of the
                        temporary variable, and the result is input to the
                        the hash algorithm H to produce a digest value, and
                        the temporary variable is set to this digest value;
                      - the value of the temporary variable is XOR-ed with
                        the first (next) L-octets (16 octets in case of MD5)
                        of the keyNew to produce the first (next) L-octets
                        (16 octets in case of MD5) of the 'delta' component.
                      - the above two steps are repeated until the unused
                        portion of the keyNew component is L octets or less,
                  - the random component is appended to the value of the
                    temporary variable, and the result is input to the

```
            hash algorithm H to produce a digest value;
        - this digest value, truncated if necessary to be the same
          length as the unused portion of the keyNew, is XOR-ed
          with the unused portion of the keyNew to produce the
          (final portion of the) 'delta' component.
```

For example, using MD5 as the hash algorithm H:

```
   iterations = (lenOfDelta - 1)/16; /* integer division */
   temp = keyOld;
   for (i = 0; i < iterations; i++) {
       temp = MD5 (temp || random);
       delta[i*16 .. (i*16)+15] =
               temp XOR keyNew[i*16 .. (i*16)+15];
   }
   temp = MD5 (temp || random);
   delta[i*16 .. lenOfDelta-1] =
           temp XOR keyNew[i*16 .. lenOfDelta-1];
```

The 'random' and 'delta' components are then concatenated as
described above, and the resulting octet string is sent to
the recipient as the new value of an instance of this object.

At the receiver side, when an instance of this object is set
to a new value, then a new value of K is computed as follows:

```
 - a temporary variable is initialized to the existing value
   of K;
 - if the length of the delta component is greater than L
   octets, then:
     - the random component is appended to the value of the
       temporary variable, and the result is input to the
       hash algorithm H to produce a digest value, and the
       temporary variable is set to this digest value;
     - the value of the temporary variable is XOR-ed with
       the first (next) L-octets (16 octets in case of MD5)
       of the delta component to produce the first (next)
       L-octets (16 octets in case of MD5) of the new value
       of K.
     - the above two steps are repeated until the unused
       portion of the delta component is L octets or less,
 - the random component is appended to the value of the
   temporary variable, and the result is input to the
   hash algorithm H to produce a digest value;
 - this digest value, truncated if necessary to be the same
   length as the unused portion of the delta component, is
   XOR-ed with the unused portion of the delta component to
   produce the (final portion of the) new value of K.
```

          For example, using MD5 as the hash algorithm H:

              iterations = (lenOfDelta - 1)/16; /* integer division */
              temp = keyOld;
              for (i = 0; i < iterations; i++) {
                  temp = MD5 (temp || random);
                  keyNew[i*16 .. (i*16)+15] =
                          temp XOR delta[i*16 .. (i*16)+15];
              }
              temp = MD5 (temp || random);
              keyNew[i*16 .. lenOfDelta-1] =
                      temp XOR delta[i*16 .. lenOfDelta-1];

          The value of an object with this syntax, whenever it is
          retrieved by the management protocol, is always the zero
          length string.

          Note that the keyOld and keyNew are the localized keys.

          Note that it is probably wise that when an SNMP entity sends
          a SetRequest to change a key, that it keeps a copy of the old
          key until it has confirmed that the key change actually
          succeeded.
          "
    SYNTAX   OCTET STRING

-- Statistics for the User-based Security Model *********************

usmStats   OBJECT IDENTIFIER ::= { usmMIBObjects 1 }

usmStatsUnsupportedSecLevels OBJECT-TYPE
    SYNTAX   Counter32
    MAX-ACCESS   read-only
    STATUS   current
    DESCRIPTION "The total number of packets received by the SNMP
                 engine which were dropped because they requested a
                 securityLevel that was unknown to the SNMP engine
                 or otherwise unavailable.
                "
    ::= { usmStats 1 }

usmStatsNotInTimeWindows OBJECT-TYPE
    SYNTAX   Counter32
    MAX-ACCESS   read-only
    STATUS   current

        DESCRIPTION "The total number of packets received by the SNMP
                     engine which were dropped because they appeared
                     outside of the authoritative SNMP engine's window.
                    "
        ::= { usmStats 2 }

usmStatsUnknownUserNames OBJECT-TYPE
        SYNTAX   Counter32
        MAX-ACCESS   read-only
        STATUS   current
        DESCRIPTION "The total number of packets received by the SNMP
                     engine which were dropped because they referenced a
                     user that was not known to the SNMP engine.
                    "
        ::= { usmStats 3 }

usmStatsUnknownEngineIDs OBJECT-TYPE
        SYNTAX   Counter32
        MAX-ACCESS   read-only
        STATUS   current
        DESCRIPTION "The total number of packets received by the SNMP
                     engine which were dropped because they referenced an
                     snmpEngineID that was not known to the SNMP engine.
                    "
        ::= { usmStats 4 }

usmStatsWrongDigests OBJECT-TYPE
        SYNTAX   Counter32
        MAX-ACCESS   read-only
        STATUS   current
        DESCRIPTION "The total number of packets received by the SNMP
                     engine which were dropped because they didn't
                     contain the expected digest value.
                    "
        ::= { usmStats 5 }

usmStatsDecryptionErrors OBJECT-TYPE
        SYNTAX   Counter32
        MAX-ACCESS   read-only
        STATUS   current
        DESCRIPTION "The total number of packets received by the SNMP
                     engine which were dropped because they could not be
                     decrypted.
                    "
        ::= { usmStats 6 }

-- The usmUser Group **

```
usmUser           OBJECT IDENTIFIER ::= { usmMIBObjects 2 }

usmUserSpinLock   OBJECT-TYPE
    SYNTAX        TestAndIncr
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION "An advisory lock used to allow several cooperating
                Command Generator Applications to coordinate their
                use of facilities to alter secrets in the
                usmUserTable.
                "
    ::= { usmUser 1 }

-- The table of valid users for the User-based Security Model ********

usmUserTable      OBJECT-TYPE
    SYNTAX        SEQUENCE OF UsmUserEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION "The table of users configured in the SNMP engine's
                Local Configuration Datastore (LCD).

                To create a new user (i.e., to instantiate a new
                conceptual row in this table), it is recommended to
                follow this procedure:

                   1)  GET(usmUserSpinLock.0) and save in sValue.
                   2)  SET(usmUserSpinLock.0=sValue,
                           usmUserCloneFrom=templateUser,
                           usmUserStatus=createAndWait)
                       You should use a template user to clone from
                       which has the proper auth/priv protocol defined.

                If the new user is to use privacy:

                   3)  generate the keyChange value based on the secret
                       privKey of the clone-from user and the secret key
                       to be used for the new user. Let us call this
                       pkcValue.
                   4)  GET(usmUserSpinLock.0) and save in sValue.
                   5)  SET(usmUserSpinLock.0=sValue,
                           usmUserPrivKeyChange=pkcValue
                           usmUserPublic=randomValue1)
                   6)  GET(usmUserPulic) and check it has randomValue1.
                       If not, repeat steps 4-6.

                If the new user will never use privacy:
```

                      7) SET(usmUserPrivProtocol=usmNoPrivProtocol)

                   If the new user is to use authentication:

                      8) generate the keyChange value based on the secret
                          authKey of the clone-from user and the secret key
                          to be used for the new user. Let us call this
                          akcValue.
                      9) GET(usmUserSpinLock.0) and save in sValue.
                      10) SET(usmUserSpinLock.0=sValue,
                             usmUserAuthKeyChange=akcValue
                             usmUserPublic=randomValue2)
                      11) GET(usmUserPulic) and check it has randomValue2.
                          If not, repeat steps 9-11.

                   If the new user will never use authentication:

                      12) SET(usmUserAuthProtocol=usmNoAuthProtocol)

                   Finally, activate the new user:

                      13) SET(usmUserStatus=active)

                   The new user should now be available and ready to be
                   used for SNMPv3 communication. Note however that access
                   to MIB data must be provided via configuration of the
                   SNMP-VIEW-BASED-ACM-MIB.

                   The use of usmUserSpinlock is to avoid conflicts with
                   another SNMP command responder application which may
                   also be acting on the usmUserTable.
                   "
       ::= { usmUser 2 }

    usmUserEntry   OBJECT-TYPE
        SYNTAX   UsmUserEntry
        MAX-ACCESS   not-accessible
        STATUS   current
        DESCRIPTION "A user configured in the SNMP engine's Local
                    Configuration Datastore (LCD) for the User-based
                    Security Model.
                    "
        INDEX   { usmUserEngineID,
                       usmUserName
                     }
        ::= { usmUserTable 1 }

    UsmUserEntry ::= SEQUENCE

```
    {
         usmUserEngineID        SnmpEngineID,
         usmUserName            SnmpAdminString,
         usmUserSecurityName    SnmpAdminString,
         usmUserCloneFrom       RowPointer,
         usmUserAuthProtocol    AutonomousType,
         usmUserAuthKeyChange   KeyChange,
         usmUserOwnAuthKeyChange KeyChange,
         usmUserPrivProtocol    AutonomousType,
         usmUserPrivKeyChange   KeyChange,
         usmUserOwnPrivKeyChange KeyChange,
         usmUserPublic          OCTET STRING,
         usmUserStorageType     StorageType,
         usmUserStatus          RowStatus
    }

usmUserEngineID  OBJECT-TYPE
    SYNTAX       SnmpEngineID
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "An SNMP engine's administratively-unique identifier.

                In a simple agent, this value is always that agent's
                own snmpEngineID value.

                The value can also take the value of the snmpEngineID
                of a remote SNMP engine with which this user can
                communicate.
                "
    ::= { usmUserEntry 1 }

usmUserName      OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(1..32))
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "A human readable string representing the name of
                the user.

                This is the (User-based Security) Model dependent
                security ID.
                "
    ::= { usmUserEntry 2 }

usmUserSecurityName OBJECT-TYPE
    SYNTAX       SnmpAdminString
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "A human readable string representing the user in
```

                    Security Model independent format.

                    The default transformation of the User-based Security
                    Model dependent security ID to the securityName and
                    vice versa is the identity function so that the
                    securityName is the same as the userName.
                    "
      ::= { usmUserEntry 3 }

usmUserCloneFrom OBJECT-TYPE
      SYNTAX   RowPointer
      MAX-ACCESS   read-create
      STATUS   current
      DESCRIPTION "A pointer to another conceptual row in this
                    usmUserTable. The user in this other conceptual
                    row is called the clone-from user.

                    When a new user is created (i.e., a new conceptual
                    row is instantiated in this table), the privacy and
                    authentication parameters of the new user must be
                    cloned from its clone-from user. These parameters are:
                      - authentication protocol (usmUserAuthProtocol)
                      - privacy protocol (usmUserPrivProtocol)
                    They will be copied regardless of what the current
                    value is.

                    Cloning also causes the initial values of the secret
                    authentication key (authKey) and the secret encryption
                    key (privKey) of the new user to be set to the same
                    value as the corresponding secret of the clone-from
                    user.

                    The first time an instance of this object is set by
                    a management operation (either at or after its
                    instantiation), the cloning process is invoked.
                    Subsequent writes are successful but invoke no
                    action to be taken by the receiver.
                    The cloning process fails with an 'inconsistentName'
                    error if the conceptual row representing the
                    clone-from user does not exist or is not in an active
                    state when the cloning process is invoked.

                    When this object is read, the ZeroDotZero OID
                    is returned.
                    "
      ::= { usmUserEntry 4 }

usmUserAuthProtocol OBJECT-TYPE

```
    SYNTAX         AutonomousType
    MAX-ACCESS     read-create
    STATUS         current
    DESCRIPTION "An indication of whether messages sent on behalf of
                 this user to/from the SNMP engine identified by
                 usmUserEngineID, can be authenticated, and if so,
                 the type of authentication protocol which is used.

                 An instance of this object is created concurrently
                 with the creation of any other object instance for
                 the same user (i.e., as part of the processing of
                 the set operation which creates the first object
                 instance in the same conceptual row).

                 If an initial set operation (i.e. at row creation time)
                 tries to set a value for an unknown or unsupported
                 protocol, then a 'wrongValue' error must be returned.

                 The value will be overwritten/set when a set operation
                 is performed on the corresponding instance of
                 usmUserCloneFrom.

                 Once instantiated, the value of such an instance of
                 this object can only be changed via a set operation to
                 the value of the usmNoAuthProtocol.

                 If a set operation tries to change the value of an
                 existing instance of this object to any value other
                 than usmNoAuthProtocol, then an 'inconsistentValue'
                 error must be returned.

                 If a set operation tries to set the value to the
                 usmNoAuthProtocol while the usmUserPrivProtocol value
                 in the same row is not equal to usmNoPrivProtocol,
                 then an 'inconsistentValue' error must be returned.
                 That means that an SNMP command generator application
                 must first ensure that the usmUserPrivProtocol is set
                 to the usmNoPrivProtocol value before it can set
                 the usmUserAuthProtocol value to usmNoAuthProtocol.
                 "
    DEFVAL       { usmNoAuthProtocol }
    ::= { usmUserEntry 5 }

usmUserAuthKeyChange OBJECT-TYPE
    SYNTAX         KeyChange  -- typically (SIZE (0 | 32)) for HMACMD5
                             -- typically (SIZE (0 | 40)) for HMACSHA
    MAX-ACCESS     read-create
    STATUS         current
```

      DESCRIPTION "An object, which when modified, causes the secret
                  authentication key used for messages sent on behalf
                  of this user to/from the SNMP engine identified by
                  usmUserEngineID, to be modified via a one-way
                  function.

                  The associated protocol is the usmUserAuthProtocol.
                  The associated secret key is the user's secret
                  authentication key (authKey). The associated hash
                  algorithm is the algorithm used by the user's
                  usmUserAuthProtocol.

                  When creating a new user, it is an 'inconsistentName'
                  error for a set operation to refer to this object
                  unless it is previously or concurrently initialized
                  through a set operation on the corresponding instance
                  of usmUserCloneFrom.

                  When the value of the corresponding usmUserAuthProtocol
                  is usmNoAuthProtocol, then a set is successful, but
                  effectively is a no-op.

                  When this object is read, the zero-length (empty)
                  string is returned.

                  The recommended way to do a key change is as follows:

                    1) GET(usmUserSpinLock.0) and save in sValue.
                    2) generate the keyChange value based on the old
                       (existing) secret key and the new secret key,
                       let us call this kcValue.

                  If you do the key change on behalf of another user:

                    3) SET(usmUserSpinLock.0=sValue,
                           usmUserAuthKeyChange=kcValue
                           usmUserPublic=randomValue)

                  If you do the key change for yourself:

                    4) SET(usmUserSpinLock.0=sValue,
                           usmUserOwnAuthKeyChange=kcValue
                           usmUserPublic=randomValue)

                  If you get a response with error-status of noError,
                  then the SET succeeded and the new key is active.
                  If you do not get a response, then you can issue a
                  GET(usmUserPublic) and check if the value is equal

```
                  to the randomValue you did send in the SET. If so, then
                  the key change succeeded and the new key is active
                  (probably the response got lost). If not, then the SET
                  request probably never reached the target and so you
                  can start over with the procedure above.
                  "
    DEFVAL      { ''H }    -- the empty string
    ::= { usmUserEntry 6 }

usmUserOwnAuthKeyChange OBJECT-TYPE
    SYNTAX       KeyChange   -- typically (SIZE (0 | 32)) for HMACMD5
                             -- typically (SIZE (0 | 40)) for HMACSHA
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "Behaves exactly as usmUserAuthKeyChange, with one
                  notable difference: in order for the set operation
                  to succeed, the usmUserName of the operation
                  requester must match the usmUserName that
                  indexes the row which is targeted by this
                  operation.
                  In addition, the USM security model must be
                  used for this operation.

                  The idea here is that access to this column can be
                  public, since it will only allow a user to change
                  his own secret authentication key (authKey).
                  Note that this can only be done once the row is active.

                  When a set is received and the usmUserName of the
                  requester is not the same as the umsUserName that
                  indexes the row which is targeted by this operation,
                  then a 'noAccess' error must be returned.

                  When a set is received and the security model in use
                  is not USM, then a 'noAccess' error must be returned.
                  "
    DEFVAL      { ''H }    -- the empty string
    ::= { usmUserEntry 7 }

usmUserPrivProtocol OBJECT-TYPE
    SYNTAX       AutonomousType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "An indication of whether messages sent on behalf of
                  this user to/from the SNMP engine identified by
                  usmUserEngineID, can be protected from disclosure,
                  and if so, the type of privacy protocol which is used.
```

                      An instance of this object is created concurrently
                      with the creation of any other object instance for
                      the same user (i.e., as part of the processing of
                      the set operation which creates the first object
                      instance in the same conceptual row).

                      If an initial set operation (i.e. at row creation time)
                      tries to set a value for an unknown or unsupported
                      protocol, then a 'wrongValue' error must be returned.

                      The value will be overwritten/set when a set operation
                      is performed on the corresponding instance of
                      usmUserCloneFrom.

                      Once instantiated, the value of such an instance of
                      this object can only be changed via a set operation to
                      the value of the usmNoPrivProtocol.

                      If a set operation tries to change the value of an
                      existing instance of this object to any value other
                      than usmNoPrivProtocol, then an 'inconsistentValue'
                      error must be returned.

                      Note that if any privacy protocol is used, then you
                      must also use an authentication protocol. In other
                      words, if usmUserPrivProtocol is set to anything else
                      than usmNoPrivProtocol, then the corresponding instance
                      of usmUserAuthProtocol cannot have a value of
                      usmNoAuthProtocol. If it does, then an
                      'inconsistentValue' error must be returned.
                      "
        DEFVAL   { usmNoPrivProtocol }
        ::= { usmUserEntry 8 }

usmUserPrivKeyChange OBJECT-TYPE
        SYNTAX   KeyChange -- typically (SIZE (0 | 32)) for DES
        MAX-ACCESS   read-create
        STATUS   current
        DESCRIPTION "An object, which when modified, causes the secret
                      encryption key used for messages sent on behalf
                      of this user to/from the SNMP engine identified by
                      usmUserEngineID, to be modified via a one-way
                      function.

                      The associated protocol is the usmUserPrivProtocol.
                      The associated secret key is the user's secret
                      privacy key (privKey). The associated hash
                      algorithm is the algorithm used by the user's

                    usmUserAuthProtocol.

                    When creating a new user, it is an 'inconsistentName'
                    error for a set operation to refer to this object
                    unless it is previously or concurrently initialized
                    through a set operation on the corresponding instance
                    of usmUserCloneFrom.

                    When the value of the corresponding usmUserPrivProtocol
                    is usmNoPrivProtocol, then a set is successful, but
                    effectively is a no-op.

                    When this object is read, the zero-length (empty)
                    string is returned.
                    See the description clause of usmUserAuthKeyChange for
                    a recommended procedure to do a key change.
                    "
    DEFVAL   { ''H }   -- the empty string
    ::= { usmUserEntry 9 }

usmUserOwnPrivKeyChange OBJECT-TYPE
    SYNTAX   KeyChange -- typically (SIZE (0 | 32)) for DES
    MAX-ACCESS read-create
    STATUS   current
    DESCRIPTION "Behaves exactly as usmUserPrivKeyChange, with one
                    notable difference: in order for the Set operation
                    to succeed, the usmUserName of the operation
                    requester must match the usmUserName that indexes
                    the row which is targeted by this operation.
                    In addition, the USM security model must be
                    used for this operation.

                    The idea here is that access to this column can be
                    public, since it will only allow a user to change
                    his own secret privacy key (privKey).
                    Note that this can only be done once the row is active.

                    When a set is received and the usmUserName of the
                    requester is not the same as the umsUserName that
                    indexes the row which is targeted by this operation,
                    then a 'noAccess' error must be returned.

                    When a set is received and the security model in use
                    is not USM, then a 'noAccess' error must be returned.
                    "
    DEFVAL   { ''H }   -- the empty string
    ::= { usmUserEntry 10 }

usmUserPublic   OBJECT-TYPE
    SYNTAX   OCTET STRING (SIZE(0..32))
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION "A publicly-readable value which can be written as part
                 of the procedure for changing a user's secret
                 authentication and/or privacy key, and later read to
                 determine whether the change of the secret was
                 effected.
                "
    DEFVAL   { ''H } -- the empty string
    ::= { usmUserEntry 11 }

usmUserStorageType OBJECT-TYPE
    SYNTAX   StorageType
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION "The storage type for this conceptual row.

                 Conceptual rows having the value 'permanent' must
                 allow write-access at a minimum to:

                 - usmUserAuthKeyChange, usmUserOwnAuthKeyChange
                   and usmUserPublic for a user who employs
                   authentication, and
                 - usmUserPrivKeyChange, usmUserOwnPrivKeyChange
                   and usmUserPublic for a user who employs
                   privacy.

                 Note that any user who employs authentication or
                 privacy must allow its secret(s) to be updated and
                 thus cannot be 'readOnly'.

                 If an initial set operation tries to set the value to
                 'readOnly' for a user who employs authentication or
                 privacy, then an 'inconsistentValue' error must be
                 returned. Note that if the value has been previously
                 set (implicit or explicit) to any value, then the rules
                 as defined in the StorageType Textual Convention apply.

                 It is an implementation issue to decide if a SET for
                 a readOnly or permanent row is accepted at all. In some
                 contexts this may make sense, in others it may not. If
                 a SET for a readOnly or permanent row is not accepted
                 at all, then a 'wrongValue' error must be returned.
                "
    DEFVAL   { nonVolatile }
    ::= { usmUserEntry 12 }

```
usmUserStatus     OBJECT-TYPE
    SYNTAX        RowStatus
    MAX-ACCESS    read-create
    STATUS        current
    DESCRIPTION "The status of this conceptual row.

                Until instances of all corresponding columns are
                appropriately configured, the value of the
                corresponding instance of the usmUserStatus column
                is 'notReady'.

                In particular, a newly created row for a user who
                employs authentication, cannot be made active until the
                corresponding usmUserCloneFrom and usmUserAuthKeyChange
                have been set.

                Further, a newly created row for a user who also
                employs privacy, cannot be made active until the
                usmUserPrivKeyChange has been set.

                The RowStatus TC [RFC2579] requires that this
                DESCRIPTION clause states under which circumstances
                other objects in this row can be modified:

                The value of this object has no effect on whether
                other objects in this conceptual row can be modified,
                except for usmUserOwnAuthKeyChange and
                usmUserOwnPrivKeyChange. For these 2 objects, the
                value of usmUserStatus MUST be active.
                "
    ::= { usmUserEntry 13 }

-- Conformance Information ******************************************

usmMIBCompliances OBJECT IDENTIFIER ::= { usmMIBConformance 1 }
usmMIBGroups      OBJECT IDENTIFIER ::= { usmMIBConformance 2 }

-- Compliance statements

usmMIBCompliance MODULE-COMPLIANCE
    STATUS        current
    DESCRIPTION "The compliance statement for SNMP engines which
                 implement the SNMP-USER-BASED-SM-MIB.
                "

    MODULE        -- this module
        MANDATORY-GROUPS { usmMIBBasicGroup }
```

```
         OBJECT          usmUserAuthProtocol
         MIN-ACCESS      read-only
         DESCRIPTION     "Write access is not required."

         OBJECT          usmUserPrivProtocol
         MIN-ACCESS      read-only
         DESCRIPTION     "Write access is not required."

     ::= { usmMIBCompliances 1 }

-- Units of compliance
usmMIBBasicGroup OBJECT-GROUP
    OBJECTS     {
                  usmStatsUnsupportedSecLevels,
                  usmStatsNotInTimeWindows,
                  usmStatsUnknownUserNames,
                  usmStatsUnknownEngineIDs,
                  usmStatsWrongDigests,
                  usmStatsDecryptionErrors,
                  usmUserSpinLock,
                  usmUserSecurityName,
                  usmUserCloneFrom,
                  usmUserAuthProtocol,
                  usmUserAuthKeyChange,
                  usmUserOwnAuthKeyChange,
                  usmUserPrivProtocol,
                  usmUserPrivKeyChange,
                  usmUserOwnPrivKeyChange,
                  usmUserPublic,
                  usmUserStorageType,
                  usmUserStatus
                }
    STATUS      current
    DESCRIPTION "A collection of objects providing for configuration
                 of an SNMP engine which implements the SNMP
                 User-based Security Model.
                "
    ::= { usmMIBGroups 1 }

END
```

6. HMAC-MD5-96 Authentication Protocol

   This section describes the HMAC-MD5-96 authentication protocol. This
   authentication protocol is the first defined for the User-based
   Security Model. It uses MD5 hash-function which is described in
   [MD5], in HMAC mode described in [RFC2104], truncating the output to
   96 bits.

   This protocol is identified by usmHMACMD5AuthProtocol.

   Over time, other authentication protocols may be defined either as a
   replacement of this protocol or in addition to this protocol.

6.1. Mechanisms

   - In support of data integrity, a message digest algorithm is
     required. A digest is calculated over an appropriate portion of an
     SNMP message and included as part of the message sent to the
     recipient.

   - In support of data origin authentication and data integrity,
     a secret value is prepended to SNMP message prior to computing the
     digest; the calculated digest is partially inserted into the SNMP
     message prior to transmission, and the prepended value is not
     transmitted. The secret value is shared by all SNMP engines
     authorized to originate messages on behalf of the appropriate user.

6.1.1. Digest Authentication Mechanism

   The Digest Authentication Mechanism defined in this memo provides
   for:

   - verification of the integrity of a received message, i.e., the
     message received is the message sent.

     The integrity of the message is protected by computing a digest
     over an appropriate portion of the message. The digest is computed
     by the originator of the message, transmitted with the message, and
     verified by the recipient of the message.

   - verification of the user on whose behalf the message was generated.

     A secret value known only to SNMP engines authorized to generate
     messages on behalf of a user is used in HMAC mode (see [RFC2104]).
     It also recommends the hash-function output used as Message
     Authentication Code, to be truncated.

This protocol uses the MD5 [MD5] message digest algorithm. A 128-bit
MD5 digest is calculated in a special (HMAC) way over the designated
portion of an SNMP message and the first 96 bits of this digest is
included as part of the message sent to the recipient. The size of
the digest carried in a message is 12 octets. The size of the private
authentication key (the secret) is 16 octets. For the details see
section 6.3.

6.2. Elements of the Digest Authentication Protocol

   This section contains definitions required to realize the
   authentication module defined in this section of this memo.

6.2.1. Users

   Authentication using this authentication protocol makes use of a
   defined set of userNames. For any user on whose behalf a message must
   be authenticated at a particular SNMP engine, that SNMP engine must
   have knowledge of that user. An SNMP engine that wishes to
   communicate with another SNMP engine must also have knowledge of a
   user known to that engine, including knowledge of the applicable
   attributes of that user.

   A user and its attributes are defined as follows:

   <userName>
     A string representing the name of the user.
   <authKey>
     A user's secret key to be used when calculating a digest.
     It MUST be 16 octets long for MD5.

6.2.2. msgAuthoritativeEngineID

   The msgAuthoritativeEngineID value contained in an authenticated
   message specifies the authoritative SNMP engine for that particular
   message (see the definition of SnmpEngineID in the SNMP Architecture
   document [RFC2571]).

   The user's (private) authentication key is normally different at each
   authoritative SNMP engine and so the snmpEngineID is used to select
   the proper key for the authentication process.

6.2.3. SNMP Messages Using this Authentication Protocol

   Messages using this authentication protocol carry a
   msgAuthenticationParameters field as part of the
   msgSecurityParameters. For this protocol, the

   msgAuthenticationParameters field is the serialized OCTET STRING
   representing the first 12 octets of the HMAC-MD5-96 output done over
   the wholeMsg.

   The digest is calculated over the wholeMsg so if a message is
   authenticated, that also means that all the fields in the message are
   intact and have not been tampered with.

6.2.4. Services provided by the HMAC-MD5-96 Authentication Module

   This section describes the inputs and outputs that the HMAC-MD5-96
   Authentication module expects and produces when the User-based
   Security module calls the HMAC-MD5-96 Authentication module for
   services.

6.2.4.1. Services for Generating an Outgoing SNMP Message

   The HMAC-MD5-96 authentication protocol assumes that the selection of
   the authKey is done by the caller and that the caller passes the
   secret key to be used.

   Upon completion the authentication module returns statusInformation
   and, if the message digest was correctly calculated, the wholeMsg
   with the digest inserted at the proper place. The abstract service
   primitive is:

   statusInformation =   -- success or failure
     authenticateOutgoingMsg(
     IN   authKey   -- secret key for authentication
     IN   wholeMsg   -- unauthenticated complete message
     OUT   authenticatedWholeMsg   -- complete authenticated message
)

   The abstract data elements are:

     statusInformation
       An indication of whether the authentication process was
       successful. If not it is an indication of the problem.
     authKey
       The secret key to be used by the authentication algorithm.
       The length of this key MUST be 16 octets.
     wholeMsg
       The message to be authenticated.
     authenticatedWholeMsg
       The authenticated message (including inserted digest) on output.

      Note, that authParameters field is filled by the authentication
      module and this module and this field should be already present in
      the wholeMsg before the Message Authentication Code (MAC) is
      generated.

6.2.4.2. Services for Processing an Incoming SNMP Message

      The HMAC-MD5-96 authentication protocol assumes that the selection of
      the authKey is done by the caller and that the caller passes the
      secret key to be used.

      Upon completion the authentication module returns statusInformation
      and, if the message digest was correctly calculated, the wholeMsg as
      it was processed. The abstract service primitive is:

      statusInformation =   -- success or failure
        authenticateIncomingMsg(
        IN   authKey   -- secret key for authentication
        IN   authParameters   -- as received on the wire
        IN   wholeMsg   -- as received on the wire
        OUT   authenticatedWholeMsg   -- complete authenticated message
)

      The abstract data elements are:

        statusInformation
          An indication of whether the authentication process was
          successful. If not it is an indication of the problem.
        authKey
          The secret key to be used by the authentication algorithm.
          The length of this key MUST be 16 octets.
        authParameters
          The authParameters from the incoming message.
        wholeMsg
          The message to be authenticated on input and the authenticated
          message on output.
        authenticatedWholeMsg
          The whole message after the authentication check is complete.

6.3. Elements of Procedure

      This section describes the procedures for the HMAC-MD5-96
      authentication protocol.

6.3.1. Processing an Outgoing Message

   This section describes the procedure followed by an SNMP engine
   whenever it must authenticate an outgoing message using the
   usmHMACMD5AuthProtocol.

   1) The msgAuthenticationParameters field is set to the serialization,
      according to the rules in [RFC1906], of an OCTET STRING containing
      12 zero octets.

   2) From the secret authKey, two keys K1 and K2 are derived:

         a) extend the authKey to 64 octets by appending 48 zero
            octets; save it as extendedAuthKey
         b) obtain IPAD by replicating the octet 0x36 64 times;
         c) obtain K1 by XORing extendedAuthKey with IPAD;
         d) obtain OPAD by replicating the octet 0x5C 64 times;
         e) obtain K2 by XORing extendedAuthKey with OPAD.

   3) Prepend K1 to the wholeMsg and calculate MD5 digest over it
      according to [MD5].

   4) Prepend K2 to the result of the step 4 and calculate MD5 digest
      over it according to [MD5]. Take the first 12 octets of the final
      digest – this is Message Authentication Code (MAC).

   5) Replace the msgAuthenticationParameters field with MAC obtained
      in the step 4.

   6) The authenticatedWholeMsg is then returned to the caller
      together with statusInformation indicating success.

6.3.2. Processing an Incoming Message

   This section describes the procedure followed by an SNMP engine
   whenever it must authenticate an incoming message using the
   usmHMACMD5AuthProtocol.

   1) If the digest received in the msgAuthenticationParameters field
       is not 12 octets long, then an failure and an errorIndication
       (authenticationError) is returned to the calling module.

   2) The MAC received in the msgAuthenticationParameters field
       is saved.

   3) The digest in the msgAuthenticationParameters field is replaced
       by the 12 zero octets.

      4) From the secret authKey, two keys K1 and K2 are derived:

           a) extend the authKey to 64 octets by appending 48 zero
              octets; save it as extendedAuthKey
           b) obtain IPAD by replicating the octet 0x36 64 times;
           c) obtain K1 by XORing extendedAuthKey with IPAD;
           d) obtain OPAD by replicating the octet 0x5C 64 times;
           e) obtain K2 by XORing extendedAuthKey with OPAD.

      5) The MAC is calculated over the wholeMsg:

           a) prepend K1 to the wholeMsg and calculate the MD5 digest
              over it;
           b) prepend K2 to the result of step 5.a and calculate the
              MD5 digest over it;
           c) first 12 octets of the result of step 5.b is the MAC.

         The msgAuthenticationParameters field is replaced with the MAC
         value that was saved in step 2.

      6) Then the newly calculated MAC is compared with the MAC
          saved in step 2. If they do not match, then an failure and an
          errorIndication (authenticationFailure) is returned to the
          calling module.

      7) The authenticatedWholeMsg and statusInformation indicating
          success are then returned to the caller.

7. HMAC-SHA-96 Authentication Protocol

   This section describes the HMAC-SHA-96 authentication protocol. This
   protocol uses the SHA hash-function which is described in [SHA-NIST],
   in HMAC mode described in [RFC2104], truncating the output to 96
   bits.

   This protocol is identified by usmHMACSHAAuthProtocol.

   Over time, other authentication protocols may be defined either as a
   replacement of this protocol or in addition to this protocol.

7.1. Mechanisms

   - In support of data integrity, a message digest algorithm is
     required. A digest is calculated over an appropriate portion of an
     SNMP message and included as part of the message sent to the
     recipient.

   - In support of data origin authentication and data integrity,
     a secret value is prepended to the SNMP message prior to computing
     the digest; the calculated digest is then partially inserted into
     the message prior to transmission. The prepended secret is not
     transmitted. The secret value is shared by all SNMP engines
     authorized to originate messages on behalf of the appropriate user.

7.1.1. Digest Authentication Mechanism

   The Digest Authentication Mechanism defined in this memo provides
   for:

   - verification of the integrity of a received message, i.e., the
     the message received is the message sent.

     The integrity of the message is protected by computing a digest
     over an appropriate portion of the message. The digest is computed
     by the originator of the message, transmitted with the message, and
     verified by the recipient of the message.

   - verification of the user on whose behalf the message was generated.

     A secret value known only to SNMP engines authorized to generate
     messages on behalf of a user is used in HMAC mode (see [RFC2104]).
     It also recommends the hash-function output used as Message
     Authentication Code, to be truncated.

   This mechanism uses the SHA [SHA-NIST] message digest algorithm. A
   160-bit SHA digest is calculated in a special (HMAC) way over the
   designated portion of an SNMP message and the first 96 bits of this
   digest is included as part of the message sent to the recipient. The
   size of the digest carried in a message is 12 octets. The size of the
   private authentication key (the secret) is 20 octets. For the details
   see section 7.3.

7.2. Elements of the HMAC-SHA-96 Authentication Protocol

   This section contains definitions required to realize the
   authentication module defined in this section of this memo.

7.2.1. Users

   Authentication using this authentication protocol makes use of a
   defined set of userNames. For any user on whose behalf a message
   must be authenticated at a particular SNMP engine, that SNMP engine
   must have knowledge of that user. An SNMP engine that wishes to

communicate with another SNMP engine must also have knowledge of a
user known to that engine, including knowledge of the applicable
attributes of that user.

A user and its attributes are defined as follows:

<userName>
  A string representing the name of the user.
<authKey>
  A user's secret key to be used when calculating a digest.
  It MUST be 20 octets long for SHA.

7.2.2. msgAuthoritativeEngineID

The msgAuthoritativeEngineID value contained in an authenticated
message specifies the authoritative SNMP engine for that particular
message (see the definition of SnmpEngineID in the SNMP Architecture
document [RFC2571]).

The user's (private) authentication key is normally different at each
authoritative SNMP engine and so the snmpEngineID is used to select
the proper key for the authentication process.

7.2.3. SNMP Messages Using this Authentication Protocol

Messages using this authentication protocol carry a
msgAuthenticationParameters field as part of the
msgSecurityParameters. For this protocol, the
msgAuthenticationParameters field is the serialized OCTET STRING
representing the first 12 octets of HMAC-SHA-96 output done over the
wholeMsg.

The digest is calculated over the wholeMsg so if a message is
authenticated, that also means that all the fields in the message are
intact and have not been tampered with.

7.2.4. Services provided by the HMAC-SHA-96 Authentication Module

This section describes the inputs and outputs that the HMAC-SHA-96
Authentication module expects and produces when the User-based
Security module calls the HMAC-SHA-96 Authentication module for
services.

7.2.4.1. Services for Generating an Outgoing SNMP Message

HMAC-SHA-96 authentication protocol assumes that the selection of the
authKey is done by the caller and that the caller passes the secret
key to be used.

   Upon completion the authentication module returns statusInformation
   and, if the message digest was correctly calculated, the wholeMsg
   with the digest inserted at the proper place. The abstract service
   primitive is:

   statusInformation =   -- success or failure
     authenticateOutgoingMsg(
     IN   authKey   -- secret key for authentication
     IN   wholeMsg   -- unauthenticated complete message
     OUT   authenticatedWholeMsg   -- complete authenticated message
)

   The abstract data elements are:

     statusInformation
       An indication of whether the authentication process was
       successful. If not it is an indication of the problem.
     authKey
       The secret key to be used by the authentication algorithm.
       The length of this key MUST be 20 octets.
     wholeMsg
       The message to be authenticated.
     authenticatedWholeMsg
       The authenticated message (including inserted digest) on output.

   Note, that authParameters field is filled by the authentication
   module and this field should be already present in the wholeMsg
   before the Message Authentication Code (MAC) is generated.

7.2.4.2. Services for Processing an Incoming SNMP Message

   HMAC-SHA-96 authentication protocol assumes that the selection of the
   authKey is done by the caller and that the caller passes the secret
   key to be used.

   Upon completion the authentication module returns statusInformation
   and, if the message digest was correctly calculated, the wholeMsg as
   it was processed. The abstract service primitive is:

   statusInformation =   -- success or failure
     authenticateIncomingMsg(
     IN   authKey   -- secret key for authentication
     IN   authParameters   -- as received on the wire
     IN   wholeMsg   -- as received on the wire
     OUT   authenticatedWholeMsg   -- complete authenticated message
)

The abstract data elements are:

  statusInformation
    An indication of whether the authentication process was
    successful. If not it is an indication of the problem.
  authKey
    The secret key to be used by the authentication algorithm.
    The length of this key MUST be 20 octets.
  authParameters
    The authParameters from the incoming message.
  wholeMsg
    The message to be authenticated on input and the authenticated
    message on output.
  authenticatedWholeMsg
    The whole message after the authentication check is complete.

7.3. Elements of Procedure

This section describes the procedures for the HMAC-SHA-96
authentication protocol.

7.3.1. Processing an Outgoing Message

This section describes the procedure followed by an SNMP engine
whenever it must authenticate an outgoing message using the
usmHMACSHAAuthProtocol.

1) The msgAuthenticationParameters field is set to the
   serialization, according to the rules in [RFC1906], of an OCTET
   STRING containing 12 zero octets.

2) From the secret authKey, two keys K1 and K2 are derived:

    a) extend the authKey to 64 octets by appending 44 zero
       octets; save it as extendedAuthKey
    b) obtain IPAD by replicating the octet 0x36 64 times;
    c) obtain K1 by XORing extendedAuthKey with IPAD;
    d) obtain OPAD by replicating the octet 0x5C 64 times;
    e) obtain K2 by XORing extendedAuthKey with OPAD.

3) Prepend K1 to the wholeMsg and calculate the SHA digest over it
   according to [SHA-NIST].

4) Prepend K2 to the result of the step 4 and calculate SHA digest
   over it according to [SHA-NIST]. Take the first 12 octets of the
   final digest - this is Message Authentication Code (MAC).

5) Replace the msgAuthenticationParameters field with MAC obtained
   in the step 5.

6) The authenticatedWholeMsg is then returned to the caller
   together with statusInformation indicating success.

7.3.2. Processing an Incoming Message

   This section describes the procedure followed by an SNMP engine
   whenever it must authenticate an incoming message using the
   usmHMACSHAAuthProtocol.

   1) If the digest received in the msgAuthenticationParameters field
       is not 12 octets long, then an failure and an errorIndication
       (authenticationError) is returned to the calling module.

   2) The MAC received in the msgAuthenticationParameters field
       is saved.

   3) The digest in the msgAuthenticationParameters field is
       replaced by the 12 zero octets.

   4) From the secret authKey, two keys K1 and K2 are derived:

         a) extend the authKey to 64 octets by appending 44 zero
            octets; save it as extendedAuthKey
         b) obtain IPAD by replicating the octet 0x36 64 times;
         c) obtain K1 by XORing extendedAuthKey with IPAD;
         d) obtain OPAD by replicating the octet 0x5C 64 times;
         e) obtain K2 by XORing extendedAuthKey with OPAD.

   5) The MAC is calculated over the wholeMsg:

         a) prepend K1 to the wholeMsg and calculate the SHA digest
            over it;
         b) prepend K2 to the result of step 5.a and calculate the
            SHA digest over it;
         c) first 12 octets of the result of step 5.b is the MAC.

       The msgAuthenticationParameters field is replaced with the MAC
       value that was saved in step 2.

   6) The the newly calculated MAC is compared with the MAC saved in
       step 2. If they do not match, then a failure and an
       errorIndication (authenticationFailure) are returned to the
       calling module.

   7) The authenticatedWholeMsg and statusInformation indicating
       success are then returned to the caller.

8. CBC-DES Symmetric Encryption Protocol

   This section describes the CBC-DES Symmetric Encryption Protocol.
   This protocol is the first privacy protocol defined for the User-
   based Security Model.

   This protocol is identified by usmDESPrivProtocol.

   Over time, other privacy protocols may be defined either as a
   replacement of this protocol or in addition to this protocol.

8.1. Mechanisms

   - In support of data confidentiality, an encryption algorithm is
     required. An appropriate portion of the message is encrypted prior
     to being transmitted. The User-based Security Model specifies that
     the scopedPDU is the portion of the message that needs to be
     encrypted.

   - A secret value in combination with a timeliness value is used
     to create the en/decryption key and the initialization vector. The
     secret value is shared by all SNMP engines authorized to originate
     messages on behalf of the appropriate user.

8.1.1. Symmetric Encryption Protocol

   The Symmetric Encryption Protocol defined in this memo provides
   support for data confidentiality. The designated portion of an SNMP
   message is encrypted and included as part of the message sent to the
   recipient.

   Two organizations have published specifications defining the DES:
   the National Institute of Standards and Technology (NIST) [DES-NIST]
   and the American National Standards Institute [DES-ANSI]. There is a
   companion Modes of Operation specification for each definition
   ([DESO-NIST] and [DESO-ANSI], respectively).

   The NIST has published three additional documents that implementors
   may find useful.

   - There is a document with guidelines for implementing and using
     the DES, including functional specifications for the DES and its
     modes of operation [DESG-NIST].

    - There is a specification of a validation test suite for the DES
      [DEST-NIST]. The suite is designed to test all aspects of the DES
      and is useful for pinpointing specific problems.

    - There is a specification of a maintenance test for the DES
      [DESM-NIST]. The test utilizes a minimal amount of data and
      processing to test all components of the DES. It provides a simple
      yes-or-no indication of correct operation and is useful to run as
      part of an initialization step, e.g., when a computer re-boots.

8.1.1.1. DES key and Initialization Vector.

   The first 8 octets of the 16-octet secret (private privacy key) are
   used as a DES key. Since DES uses only 56 bits, the Least
   Significant Bit in each octet is disregarded.

   The Initialization Vector for encryption is obtained using the
   following procedure.

   The last 8 octets of the 16-octet secret (private privacy key) are
   used as pre-IV.

   In order to ensure that the IV for two different packets encrypted by
   the same key, are not the same (i.e., the IV does not repeat) we need
   to "salt" the pre-IV with something unique per packet. An 8-octet
   string is used as the "salt". The concatenation of the generating
   SNMP engine's 32-bit snmpEngineBoots and a local 32-bit integer, that
   the encryption engine maintains, is input to the "salt". The 32-bit
   integer is initialized to an arbitrary value at boot time.

   The 32-bit snmpEngineBoots is converted to the first 4 octets (Most
   Significant Byte first) of our "salt". The 32-bit integer is then
   converted to the last 4 octet (Most Significant Byte first) of our
   "salt". The resulting "salt" is then XOR-ed with the pre-IV to
   obtain the IV. The 8-octet "salt" is then put into the
   privParameters field encoded as an OCTET STRING. The "salt" integer
   is then modified. We recommend that it be incremented by one and
   wrap when it reaches the maximum value.

   How exactly the value of the "salt" (and thus of the IV) varies, is
   an implementation issue, as long as the measures are taken to avoid
   producing a duplicate IV.

   The "salt" must be placed in the privParameters field to enable the
   receiving entity to compute the correct IV and to decrypt the
   message.

8.1.1.2. Data Encryption.

   The data to be encrypted is treated as sequence of octets. Its length
   should be an integral multiple of 8 - and if it is not, the data is
   padded at the end as necessary. The actual pad value is irrelevant.

   The data is encrypted in Cipher Block Chaining mode.

   The plaintext is divided into 64-bit blocks.

   The plaintext for each block is XOR-ed with the ciphertext of the
   previous block, the result is encrypted and the output of the
   encryption is the ciphertext for the block. This procedure is
   repeated until there are no more plaintext blocks.

   For the very first block, the Initialization Vector is used instead
   of the ciphertext of the previous block.

8.1.1.3. Data Decryption

   Before decryption, the encrypted data length is verified. If the
   length of the OCTET STRING to be decrypted is not an integral
   multiple of 8 octets, the decryption process is halted and an
   appropriate exception noted. When decrypting, the padding is
   ignored.

   The first ciphertext block is decrypted, the decryption output is
   XOR-ed with the Initialization Vector, and the result is the first
   plaintext block.

   For each subsequent block, the ciphertext block is decrypted, the
   decryption output is XOR-ed with the previous ciphertext block and
   the result is the plaintext block.

8.2. Elements of the DES Privacy Protocol

   This section contains definitions required to realize the privacy
   module defined by this memo.

8.2.1. Users

   Data en/decryption using this Symmetric Encryption Protocol makes use
   of a defined set of userNames. For any user on whose behalf a
   message must be en/decrypted at a particular SNMP engine, that SNMP
   engine must have knowledge of that user. An SNMP engine that wishes
   to communicate with another SNMP engine must also have knowledge of a
   user known to that SNMP engine, including knowledge of the applicable
   attributes of that user.

A user and its attributes are defined as follows:

<userName>
  An octet string representing the name of the user.

<privKey>
  A user's secret key to be used as input for the DES key and IV.
  The length of this key MUST be 16 octets.

8.2.2. msgAuthoritativeEngineID

   The msgAuthoritativeEngineID value contained in an authenticated
   message specifies the authoritative SNMP engine for that particular
   message (see the definition of SnmpEngineID in the SNMP Architecture
   document [RFC2571]).

   The user's (private) privacy key is normally different at each
   authoritative SNMP engine and so the snmpEngineID is used to select
   the proper key for the en/decryption process.

8.2.3. SNMP Messages Using this Privacy Protocol

   Messages using this privacy protocol carry a msgPrivacyParameters
   field as part of the msgSecurityParameters. For this protocol, the
   msgPrivacyParameters field is the serialized OCTET STRING
   representing the "salt" that was used to create the IV.

8.2.4. Services provided by the DES Privacy Module

   This section describes the inputs and outputs that the DES Privacy
   module expects and produces when the User-based Security module
   invokes the DES Privacy module for services.

8.2.4.1. Services for Encrypting Outgoing Data

   This DES privacy protocol assumes that the selection of the privKey
   is done by the caller and that the caller passes the secret key to be
   used.

   Upon completion the privacy module returns statusInformation and, if
   the encryption process was successful, the encryptedPDU and the
   msgPrivacyParameters encoded as an OCTET STRING. The abstract
   service primitive is:

   statusInformation =   -- success of failure
     encryptData(
     IN   encryptKey   -- secret key for encryption
     IN   dataToEncrypt   -- data to encrypt (scopedPDU)

```
   OUT   encryptedData              -- encrypted data (encryptedPDU)
   OUT   privParameters             -- filled in by service provider
         )
```

   The abstract data elements are:

     statusInformation
       An indication of the success or failure of the encryption
       process. In case of failure, it is an indication of the error.
     encryptKey
       The secret key to be used by the encryption algorithm.
       The length of this key MUST be 16 octets.
     dataToEncrypt
       The data that must be encrypted.
     encryptedData
       The encrypted data upon successful completion.
     privParameters
       The privParameters encoded as an OCTET STRING.

8.2.4.2. Services for Decrypting Incoming Data

   This DES privacy protocol assumes that the selection of the privKey
   is done by the caller and that the caller passes the secret key to be
   used.

   Upon completion the privacy module returns statusInformation and, if
   the decryption process was successful, the scopedPDU in plain text.
   The abstract service primitive is:

```
   statusInformation =
     decryptData(
     IN    decryptKey                -- secret key for decryption
     IN    privParameters            -- as received on the wire
     IN    encryptedData             -- encrypted data (encryptedPDU)
     OUT   decryptedData             -- decrypted data (scopedPDU)
           )
```

   The abstract data elements are:

     statusInformation
       An indication whether the data was successfully decrypted
       and if not an indication of the error.
     decryptKey
       The secret key to be used by the decryption algorithm.
       The length of this key MUST be 16 octets.
     privParameters
       The "salt" to be used to calculate the IV.

     encryptedData
        The data to be decrypted.
     decryptedData
        The decrypted data.

8.3. Elements of Procedure.

   This section describes the procedures for the DES privacy protocol.

8.3.1. Processing an Outgoing Message

   This section describes the procedure followed by an SNMP engine
   whenever it must encrypt part of an outgoing message using the
   usmDESPrivProtocol.

   1)   The secret cryptKey is used to construct the DES encryption key,
        the "salt" and the DES pre-IV (from which the IV is computed as
        described in section 8.1.1.1).

   2)   The privParameters field is set to the serialization according
        to the rules in [RFC1906] of an OCTET STRING representing the the
        "salt" string.

   3)   The scopedPDU is encrypted (as described in section 8.1.1.2)
        and the encrypted data is serialized according to the rules in
        [RFC1906] as an OCTET STRING.

   4)   The serialized OCTET STRING representing the encrypted
        scopedPDU together with the privParameters and statusInformation
        indicating success is returned to the calling module.

8.3.2. Processing an Incoming Message

   This section describes the procedure followed by an SNMP engine
   whenever it must decrypt part of an incoming message using the
   usmDESPrivProtocol.

   1)   If the privParameters field is not an 8-octet OCTET STRING,
        then an error indication (decryptionError) is returned to the
        calling module.

   2)   The "salt" is extracted from the privParameters field.

   3)   The secret cryptKey and the "salt" are then used to construct the
        DES decryption key and pre-IV (from which the IV is computed as
        described in section 8.1.1.1).

   4) The encryptedPDU is then decrypted (as described in
       section 8.1.1.3).

   5) If the encryptedPDU cannot be decrypted, then an error
       indication (decryptionError) is returned to the calling module.

   6) The decrypted scopedPDU and statusInformation indicating
       success are returned to the calling module.

9. Intellectual Property

   The IETF takes no position regarding the validity or scope of any
   intellectual property or other rights that might be claimed to
   pertain to the implementation or use of the technology described in
   this document or the extent to which any license under such rights
   might or might not be available; neither does it represent that it
   has made any effort to identify any such rights. Information on the
   IETF's procedures with respect to rights in standards-track and
   standards-related documentation can be found in BCP-11. Copies of
   claims of rights made available for publication and any assurances of
   licenses to be made available, or the result of an attempt made to
   obtain a general license or permission for the use of such
   proprietary rights by implementors or users of this specification can
   be obtained from the IETF Secretariat.

   The IETF invites any interested party to bring to its attention any
   copyrights, patents or patent applications, or other proprietary
   rights which may cover technology that may be required to practice
   this standard. Please address the information to the IETF Executive
   Director.

10. Acknowledgements

   This document is the result of the efforts of the SNMPv3 Working
   Group. Some special thanks are in order to the following SNMPv3 WG
   members:

      Harald Tveit Alvestrand (Maxware)
      Dave Battle (SNMP Research, Inc.)
      Alan Beard (Disney Worldwide Services)
      Paul Berrevoets (SWI Systemware/Halcyon Inc.)
      Martin Bjorklund (Ericsson)
      Uri Blumenthal (IBM T.J. Watson Research Center)
      Jeff Case (SNMP Research, Inc.)
      John Curran (BBN)
      Mike Daniele (Compaq Computer Corporation))
      T. Max Devlin (Eltrax Systems)
      John Flick (Hewlett Packard)

        Keith McCloghrie (Cisco Systems)
        Brian O'Keefe (Hewlett Packard)
        Marshall T. Rose (Dover Beach Consulting)
        Jon Saperia (BGS Systems Inc.)
        Steve Waldbusser (International Network Services)
        Glenn W. Waters (Bell-Northern Research Ltd.)

11. Security Considerations

11.1. Recommended Practices

   This section describes practices that contribute to the secure,
   effective operation of the mechanisms defined in this memo.

   - An SNMP engine must discard SNMP Response messages that do not
     correspond to any currently outstanding Request message. It is the
     responsibility of the Message Processing module to take care of
     this. For example it can use a msgID for that.

     An SNMP Command Generator Application must discard any Response
     Class PDU for which there is no currently outstanding Confirmed
     Class PDU; for example for SNMPv2 [RFC1905] PDUs, the request-id
     component in the PDU can be used to correlate Responses to
     outstanding Requests.

     Although it would be typical for an SNMP engine and an SNMP Command
     Generator Application to do this as a matter of course, when using
     these security protocols it is significant due to the possibility
     of message duplication (malicious or otherwise).

   - If an SNMP engine uses a msgID for correlating Response messages
     to outstanding Request messages, then it MUST use different msgIDs
     in all such Request messages that it sends out during a Time Window
     (150 seconds) period.

     A Command Generator or Notification Originator Application MUST use
     different request-ids in all Request PDUs that it sends out during
     a TimeWindow (150 seconds) period.

     This must be done to protect against the possibility of message
     duplication (malicious or otherwise).

     For example, starting operations with a msgID and/or request-id
     value of zero is not a good idea. Initializing them with an
     unpredictable number (so they do not start out the same after each
     reboot) and then incrementing by one would be acceptable.

- An SNMP engine should perform time synchronization using
  authenticated messages in order to protect against the possibility
  of message duplication (malicious or otherwise).

- When sending state altering messages to a managed authoritative
  SNMP engine, a Command Generator Application should delay sending
  successive messages to that managed SNMP engine until a positive
  acknowledgement is received for the previous message or until the
  previous message expires.

  No message ordering is imposed by the SNMP. Messages may be
  received in any order relative to their time of generation and each
  will be processed in the ordered received. Note that when an
  authenticated message is sent to a managed SNMP engine, it will be
  valid for a period of time of approximately 150 seconds under
  normal circumstances, and is subject to replay during this period.
  Indeed, an SNMP engine and SNMP Command Generator Applications must
  cope with the loss and re-ordering of messages resulting from
  anomalies in the network as a matter of course.

  However, a managed object, snmpSetSerialNo [RFC1907], is
  specifically defined for use with SNMP Set operations in order to
  provide a mechanism to ensure that the processing of SNMP messages
  occurs in a specific order.

- The frequency with which the secrets of a User-based Security
  Model user should be changed is indirectly related to the frequency
  of their use.

  Protecting the secrets from disclosure is critical to the overall
  security of the protocols. Frequent use of a secret provides a
  continued source of data that may be useful to a cryptanalyst in
  exploiting known or perceived weaknesses in an algorithm. Frequent
  changes to the secret avoid this vulnerability.

  Changing a secret after each use is generally regarded as the most
  secure practice, but a significant amount of overhead may be
  associated with that approach.

  Note, too, in a local environment the threat of disclosure may be
  less significant, and as such the changing of secrets may be less
  frequent. However, when public data networks are used as the
  communication paths, more caution is prudent.

11.2 Defining Users

   The mechanisms defined in this document employ the notion of users on
   whose behalf messages are sent. How "users" are defined is subject
   to the security policy of the network administration. For example,
   users could be individuals (e.g., "joe" or "jane"), or a particular
   role (e.g., "operator" or "administrator"), or a combination (e.g.,
   "joe-operator", "jane-operator" or "joe-admin"). Furthermore, a user
   may be a logical entity, such as an SNMP Application or a set of SNMP
   Applications, acting on behalf of an individual or role, or set of
   individuals, or set of roles, including combinations.

   Appendix A describes an algorithm for mapping a user "password" to a
   16/20 octet value for use as either a user's authentication key or
   privacy key (or both). Note however, that using the same password
   (and therefore the same key) for both authentication and privacy is
   very poor security practice and should be strongly discouraged.
   Passwords are often generated, remembered, and input by a human.
   Human-generated passwords may be less than the 16/20 octets required
   by the authentication and privacy protocols, and brute force attacks
   can be quite easy on a relatively short ASCII character set.
   Therefore, the algorithm is Appendix A performs a transformation on
   the password. If the Appendix A algorithm is used, SNMP
   implementations (and SNMP configuration applications) must ensure
   that passwords are at least 8 characters in length. Please note that
   longer passwords with repetitive strings may result in exactly the
   same key. For example, a password 'bertbert' will result in exactly
   the same key as password 'bertbertbert'.

   Because the Appendix A algorithm uses such passwords (nearly)
   directly, it is very important that they not be easily guessed. It
   is suggested that they be composed of mixed-case alphanumeric and
   punctuation characters that don't form words or phrases that might be
   found in a dictionary. Longer passwords improve the security of the
   system. Users may wish to input multiword phrases to make their
   password string longer while ensuring that it is memorable.

   Since it is infeasible for human users to maintain different
   passwords for every SNMP engine, but security requirements strongly
   discourage having the same key for more than one SNMP engine, the
   User-based Security Model employs a compromise proposed in
   [Localized-key]. It derives the user keys for the SNMP engines from
   user's password in such a way that it is practically impossible to
   either determine the user's password, or user's key for another SNMP
   engine from any combination of user's keys on SNMP engines.

Note however, that if user's password is disclosed, then key
localization will not help and network security may be compromised in
this case. Therefore a user's password or non-localized key MUST NOT
be stored on a managed device/node. Instead the localized key SHALL
be stored (if at all) , so that, in case a device does get
compromised, no other managed or managing devices get compromised.

11.3. Conformance

To be termed a "Secure SNMP implementation" based on the User-based
Security Model, an SNMP implementation MUST:

- implement one or more Authentication Protocol(s). The HMAC-MD5-96
  and HMAC-SHA-96 Authentication Protocols defined in this memo are
  examples of such protocols.

- to the maximum extent possible, prohibit access to the secret(s)
  of each user about which it maintains information in a Local
  Configuration Datastore (LCD) under all circumstances except as
  required to generate and/or validate SNMP messages with respect to
  that user.

- implement the key-localization mechanism.

- implement the SNMP-USER-BASED-SM-MIB.

In addition, an authoritative SNMP engine SHOULD provide initial
configuration in accordance with Appendix A.1.

Implementation of a Privacy Protocol (the DES Symmetric Encryption
Protocol defined in this memo is one such protocol) is optional.

11.4. Use of Reports

The use of unsecure reports (i.e. sending them with a securityLevel
of noAuthNoPriv) potentially exposes a non-authoritative SNMP engine
to some form of attacks. Some people consider these denial of
service attacks, others don't. An installation should evaluate the
risk involved before deploying unsecure Report PDUs.

11.5 Access to the SNMP-USER-BASED-SM-MIB

The objects in this MIB may be considered sensitive in many
environments. Specifically the objects in the usmUserTable contain
information about users and their authentication and privacy
protocols. It is important to closely control (both read and write)

   access to these MIB objects by using appropriately configured Access
   Control models (for example the View-based Access Control Model as
   specified in [RFC2575]).

12. References

   [RFC1321]   Rivest, R., "Message Digest Algorithm MD5", RFC
                   1321, April 1992.

   [RFC2579]   McCloghrie, K., Perkins, D. and J. Schoenwaelder,
                   "Textual Conventions for SMIv2", STD 58, RFC 2579,
                   April 1999.

   [RFC1905]   Case, J., McCloghrie, K., Rose, M. and S. Waldbusser,
                   "Protocol Operations for Version 2 of the Simple
                   Network Management Protocol (SNMPv2)", RFC 1905,
                   January 1996.

   [RFC1906]   Case, J., McCloghrie, K., Rose, M. and S. Waldbusser,
                   "Transport Mappings for Version 2 of the Simple
                   Network Management Protocol (SNMPv2)", RFC 1906,
                   January 1996.

   [RFC1907]   Case, J., McCloghrie, K., Rose, M. and S. Waldbusser,
                   "Management Information Base for Version 2 of the
                   Simple Network Management Protocol (SNMPv2)", RFC
                   1907 January 1996.

   [RFC2104]   Krawczyk, H., Bellare, M. and R. Canetti, "HMAC:
                   Keyed-Hashing for Message Authentication", RFC 2104,
                   February 1997.

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

   [RFC2571]   Harrington, D., Presuhn, R. and B. Wijnen, "An
                   Architecture for describing SNMP Management
                   Frameworks", RFC 2571, April 1999.

   [RFC2572]   Case, J., Harrington, D., Presuhn, R. and B. Wijnen,
                   "Message Processing and Dispatching for the Simple
                   Network Management Protocol (SNMP)", RFC 2572, April
                   1999.

   [RF2575]   Wijnen, B., Presuhn, R. and K. McCloghrie, "View-
                   based Access Control Model for the Simple Network
                   Management Protocol (SNMP)", RFC 2575, April 1999.

[Localized-Key] U. Blumenthal, N. C. Hien, B. Wijnen "Key Derivation
                for Network Management Applications" IEEE Network
                Magazine, April/May issue, 1997.

[DES-NIST]   Data Encryption Standard, National Institute of
                Standards and Technology. Federal Information
                Processing Standard (FIPS) Publication 46-1.
                Supersedes FIPS Publication 46, (January, 1977;
                reaffirmed January, 1988).

[DES-ANSI]   Data Encryption Algorithm, American National
                Standards Institute. ANSI X3.92-1981, (December,
                1980).

[DESO-NIST]   DES Modes of Operation, National Institute of
                Standards and Technology. Federal Information
                Processing Standard (FIPS) Publication 81, (December,
                1980).

[DESO-ANSI]   Data Encryption Algorithm - Modes of Operation,
                American National Standards Institute. ANSI X3.106-
                1983, (May 1983).

[DESG-NIST]   Guidelines for Implementing and Using the NBS Data
                Encryption Standard, National Institute of Standards
                and Technology. Federal Information Processing
                Standard (FIPS) Publication 74, (April, 1981).

[DEST-NIST]   Validating the Correctness of Hardware
                Implementations of the NBS Data Encryption Standard,
                National Institute of Standards and Technology.
                Special Publication 500-20.

[DESM-NIST]   Maintenance Testing for the Data Encryption Standard,
                National Institute of Standards and Technology.
                Special Publication 500-61, (August, 1980).

[SHA-NIST]   Secure Hash Algorithm. NIST FIPS 180-1, (April, 1995)
                http://csrc.nist.gov/fips/fip180-1.txt (ASCII)
                http://csrc.nist.gov/fips/fip180-1.ps (Postscript)

13. Editors' Addresses

   Uri Blumenthal
   IBM T. J. Watson Research
   30 Saw Mill River Pkwy,
   Hawthorne, NY 10532
   USA

   Phone: +1-914-784-7064
   EMail: uri@watson.ibm.com

   Bert Wijnen
   IBM T. J. Watson Research
   Schagen 33
   3461 GL Linschoten
   Netherlands

   Phone: +31-348-432-794
   EMail: wijnen@vnet.ibm.com

APPENDIX A - Installation

A.1. SNMP engine Installation Parameters

   During installation, an authoritative SNMP engine SHOULD (in the
   meaning as defined in [RFC2119]) be configured with several initial
   parameters. These include:

   1) A security posture

      The choice of security posture determines if initial configuration
      is implemented and if so how. One of three possible choices is
      selected:

           minimum-secure,
           semi-secure,
           very-secure (i.e., no-initial-configuration)

      In the case of a very-secure posture, there is no initial
      configuration, and so the following steps are irrelevant.

   2) one or more secrets

      These are the authentication/privacy secrets for the first user to be
      configured.

      One way to accomplish this is to have the installer enter a
      "password" for each required secret. The password is then
      algorithmically converted into the required secret by:

      - forming a string of length 1,048,576 octets by repeating the
        value of the password as often as necessary, truncating
        accordingly, and using the resulting string as the input to the MD5
        algorithm [MD5]. The resulting digest, termed "digest1", is used
        in the next step.

      - a second string is formed by concatenating digest1, the SNMP
        engine's snmpEngineID value, and digest1. This string is used as
        input to the MD5 algorithm [MD5].

        The resulting digest is the required secret (see Appendix A.2).

      With these configured parameters, the SNMP engine instantiates the
      following usmUserEntry in the usmUserTable:

	no privacy support	privacy support
	------------------	---------------
usmUserEngineID	localEngineID	localEngineID
usmUserName	"initial"	"initial"
usmUserSecurityName	"initial"	"initial"
usmUserCloneFrom	ZeroDotZero	ZeroDotZero
usmUserAuthProtocol	usmHMACMD5AuthProtocol	usmHMACMD5AuthProtocol
usmUserAuthKeyChange	""	""
usmUserOwnAuthKeyChange	""	""
usmUserPrivProtocol	none	usmDESPrivProtocol
usmUserPrivKeyChange	""	""
usmUserOwnPrivKeyChange	""	""
usmUserPublic	""	""
usmUserStorageType	anyValidStorageType	anyValidStorageType
usmUserStatus	active	active

It is recommended to also instantiate a set of template
usmUserEntries which can be used as clone-from users for newly
created usmUserEntries. These are the two suggested entries:

	no privacy support	privacy support
	------------------	---------------
usmUserEngineID	localEngineID	localEngineID
usmUserName	"templateMD5"	"templateMD5"
usmUserSecurityName	"templateMD5"	"templateMD5"
usmUserCloneFrom	ZeroDotZero	ZeroDotZero
usmUserAuthProtocol	usmHMACMD5AuthProtocol	usmHMACMD5AuthProtocol
usmUserAuthKeyChange	""	""
usmUserOwnAuthKeyChange	""	""
usmUserPrivProtocol	none	usmDESPrivProtocol
usmUserPrivKeyChange	""	""
usmUserOwnPrivKeyChange	""	""
usmUserPublic	""	""
usmUserStorageType	permanent	permanent
usmUserStatus	active	active

```
                           no privacy support    privacy support
                           ------------------    ---------------
     usmUserEngineID       localEngineID         localEngineID
     usmUserName           "templateSHA"         "templateSHA"
     usmUserSecurityName   "templateSHA"         "templateSHA"
     usmUserCloneFrom      ZeroDotZero           ZeroDotZero
     usmUserAuthProtocol   usmHMACSHAAuthProtocol usmHMACSHAAuthProtocol
     usmUserAuthKeyChange  ""                    ""
     usmUserOwnAuthKeyChange ""                  ""
     usmUserPrivProtocol   none                  usmDESPrivProtocol
     usmUserPrivKeyChange  ""                    ""
     usmUserOwnPrivKeyChange ""                  ""
     usmUserPublic         ""                    ""
     usmUserStorageType    permanent             permanent
     usmUserStatus         active                active
```

A.2. Password to Key Algorithm

   A sample code fragment (section A.2.1) demonstrates the password to
   key algorithm which can be used when mapping a password to an
   authentication or privacy key using MD5. The reference source code
   of MD5 is available in [RFC1321].

   Another sample code fragment (section A.2.2) demonstrates the
   password to key algorithm which can be used when mapping a password
   to an authentication or privacy key using SHA (documented in
   SHA-NIST).

   An example of the results of a correct implementation is provided
   (section A.3) which an implementor can use to check if his
   implementation produces the same result.

A.2.1. Password to Key Sample Code for MD5

```
void password_to_key_md5(
   u_char *password,    /* IN */
   u_int   passwordlen, /* IN */
   u_char *engineID,    /* IN  - pointer to snmpEngineID  */
   u_int   engineLength,/* IN  - length of snmpEngineID */
   u_char *key)         /* OUT - pointer to caller 16-octet buffer */
{
   MD5_CTX     MD;
   u_char      *cp, password_buf[64];
   u_long       password_index = 0;
   u_long       count = 0, i;

   MD5Init (&MD);   /* initialize MD5 */

   /********************************************/
   /* Use while loop until we've done 1 Megabyte */
   /********************************************/
   while (count < 1048576) {
      cp = password_buf;
      for (i = 0; i < 64; i++) {
          /**********************************************/
          /* Take the next octet of the password, wrapping */
          /* to the beginning of the password as necessary.*/
          /**********************************************/
          *cp++ = password[password_index++ % passwordlen];
      }
      MD5Update (&MD, password_buf, 64);
      count += 64;
   }
   MD5Final (key, &MD);          /* tell MD5 we're done */

   /***************************************************/
   /* Now localize the key with the engineID and pass   */
   /* through MD5 to produce final key                  */
   /* May want to ensure that engineLength <= 32,       */
   /* otherwise need to use a buffer larger than 64     */
   /***************************************************/
   memcpy(password_buf, key, 16);
   memcpy(password_buf+16, engineID, engineLength);
   memcpy(password_buf+16+engineLength, key, 16);

   MD5Init(&MD);
   MD5Update(&MD, password_buf, 32+engineLength);
   MD5Final(key, &MD);
   return;
}
```

A.2.2. Password to Key Sample Code for SHA

```
void password_to_key_sha(
   u_char *password,    /* IN */
   u_int   passwordlen, /* IN */
   u_char *engineID,    /* IN  - pointer to snmpEngineID  */
   u_int   engineLength,/* IN  - length of snmpEngineID */
   u_char *key)         /* OUT - pointer to caller 20-octet buffer */
{
   SHA_CTX     SH;
   u_char      *cp, password_buf[72];
   u_long       password_index = 0;
   u_long       count = 0, i;

   SHAInit (&SH);    /* initialize SHA */

   /*********************************************/
   /* Use while loop until we've done 1 Megabyte */
   /*********************************************/
   while (count < 1048576) {
      cp = password_buf;
      for (i = 0; i < 64; i++) {
          /***********************************************/
          /* Take the next octet of the password, wrapping */
          /* to the beginning of the password as necessary.*/
          /***********************************************/
          *cp++ = password[password_index++ % passwordlen];
      }
      SHAUpdate (&SH, password_buf, 64);
      count += 64;
   }
   SHAFinal (key, &SH);           /* tell SHA we're done */

   /****************************************************/
   /* Now localize the key with the engineID and pass   */
   /* through SHA to produce final key                  */
   /* May want to ensure that engineLength <= 32,       */
   /* otherwise need to use a buffer larger than 72     */
   /****************************************************/
   memcpy(password_buf, key, 20);
   memcpy(password_buf+20, engineID, engineLength);
   memcpy(password_buf+20+engineLength, key, 20);

   SHAInit(&SH);
   SHAUpdate(&SH, password_buf, 40+engineLength);
   SHAFinal(key, &SH);
   return;
}
```

A.3.   Password to Key Sample Results

A.3.1.   Password to Key Sample Results using MD5

   The following shows a sample output of the password to key algorithm
   for a 16-octet key using MD5.

   With a password of "maplesyrup" the output of the password to key
   algorithm before the key is localized with the SNMP engine's
   snmpEngineID is:

      '9f af 32 83 88 4e 92 83 4e bc 98 47 d8 ed d9 63'H

   After the intermediate key (shown above) is localized with the
   snmpEngineID value of:

      '00 00 00 00 00 00 00 00 00 00 00 02'H

   the final output of the password to key algorithm is:

      '52 6f 5e ed 9f cc e2 6f 89 64 c2 93 07 87 d8 2b'H

A.3.2.   Password to Key Sample Results using SHA

      The following shows a sample output of the password to key
      algorithm for a 20-octet key using SHA.

      With a password of "maplesyrup" the output of the password to key
      algorithm before the key is localized with the SNMP engine's
      snmpEngineID is:

      '9f b5 cc 03 81 49 7b 37 93 52 89 39 ff 78 8d 5d 79 14 52 11'H

   After the intermediate key (shown above) is localized with the
   snmpEngineID value of:

      '00 00 00 00 00 00 00 00 00 00 00 02'H

   the final output of the password to key algorithm is:

      '66 95 fe bc 92 88 e3 62 82 23 5f c7 15 1f 12 84 97 b3 8f 3f'H

A.4. Sample encoding of msgSecurityParameters

   The msgSecurityParameters in an SNMP message are represented as an
   OCTET STRING. This OCTET STRING should be considered opaque outside a
   specific Security Model.

   The User-based Security Model defines the contents of the OCTET
   STRING as a SEQUENCE (see section 2.4).

   Given these two properties, the following is an example of the
   msgSecurityParameters for the User-based Security Model, encoded as
   an OCTET STRING:

```
     04 <length>
     30 <length>
     04 <length> <msgAuthoritativeEngineID>
     02 <length> <msgAuthoritativeEngineBoots>
     02 <length> <msgAuthoritativeEngineTime>
     04 <length> <msgUserName>
     04 0c       <HMAC-MD5-96-digest>
     04 08       <salt>
```

   Here is the example once more, but now with real values (except for
   the digest in msgAuthenticationParameters and the salt in
   msgPrivacyParameters, which depend on variable data that we have not
   defined here):

```
     Hex Data                     Description
     --------------    ----------------------------------------------
     04 39             OCTET STRING,                   length 57
     30 37             SEQUENCE,                       length 55
     04 0c 80000002    msgAuthoritativeEngineID:       IBM
           01                                          IPv4 address
           09840301                                    9.132.3.1
     02 01 01          msgAuthoritativeEngineBoots:    1
     02 02 0101        msgAuthoritativeEngineTime:     257
     04 04 62657274    msgUserName:                    bert
     04 0c 01234567    msgAuthenticationParameters:    sample value
           89abcdef
           fedcba98
     04 08 01234567    msgPrivacyParameters:           sample value
           89abcdef
```

A.5. Sample keyChange Results

A.5.1. Sample keyChange Results using MD5

   Let us assume that a user has a current password of "maplesyrup" as
   in section A.3.1. and let us also assume the snmpEngineID of 12
   octets:

      '00 00 00 00 00 00 00 00 00 00 00 02'H

   If we now want to change the password to "newsyrup", then we first
   calculate the key for the new password. It is as follows:

      '01 ad d2 73 10 7c 4e 59 6b 4b 00 f8 2b 1d 42 a7'H

   If we localize it for the above snmpEngineID, then the localized new
   key becomes:

      '87 02 1d 7b d9 d1 01 ba 05 ea 6e 3b f9 d9 bd 4a'H

   If we then use a (not so good, but easy to test) random value of:

      '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'H

   Then the value we must send for keyChange is:

      '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
       88 05 61 51 41 67 6c c9 19 61 74 e7 42 a3 25 51'H

   If this were for the privacy key, then it would be exactly the same.

A.5.2. Sample keyChange Results using SHA

   Let us assume that a user has a current password of "maplesyrup" as
   in section A.3.2. and let us also assume the snmpEngineID of 12
   octets:

      '00 00 00 00 00 00 00 00 00 00 00 02'H

   If we now want to change the password to "newsyrup", then we first
   calculate the key for the new password. It is as follows:

      '3a 51 a6 d7 36 aa 34 7b 83 dc 4a 87 e3 e5 5e e4 d6 98 ac 71'H

   If we localize it for the above snmpEngineID, then the localized new
   key becomes:

      '78 e2 dc ce 79 d5 94 03 b5 8c 1b ba a5 bf f4 63 91 f1 cd 25'H

   If we then use a (not so good, but easy to test) random value of:

      '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'H

   Then the value we must send for keyChange is:

      '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        9c 10 17 f4 fd 48 3d 2d e8 d5 fa db f8 43 92 cb 06 45 70 51'

   For the key used for privacy, the new nonlocalized key would be:

      '3a 51 a6 d7 36 aa 34 7b 83 dc 4a 87 e3 e5 5e e4 d6 98 ac 71'H

   For the key used for privacy, the new localized key would be (note
   that they localized key gets truncated to 16 octets for DES):

      '78 e2 dc ce 79 d5 94 03 b5 8c 1b ba a5 bf f4 63'H

   If we then use a (not so good, but easy to test) random value of:

      '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'H

   Then the value we must send for keyChange for the privacy key is:

      '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
       '7e f8 d8 a4 c9 cd b2 6b 47 59 1c d8 52 ff 88 b5'H

B. Change Log

   Changes made since RFC2274:
   - Fixed msgUserName to allow size of zero and explain that this can
   be used for snmpEngineID discovery.
   - Clarified section 3.1 steps 4.b, 5, 6 and 8.b.
   - Clarified section 3.2 paragraph 2.
   - Clarified section 3.2 step 7.a last paragraph, step 7.b.1 second
   bullet and step 7.b.2 third bullet.
   - Clarified section 4 to indicate that discovery can use a userName
   of zero length in unAuthenticated messages, whereas a valid userName
   must be used in authenticated messages.
   - Added REVISION clauses to MODULE-IDENTITY
   - Clarified KeyChange TC by adding a note that localized keys must be
   used when calculating a KeyChange value.
   - Added clarifying text to the DESCRIPTION clause of usmUserTable.
   Added text describes a recommended procedure for adding a new user.
   - Clarified the use of usmUserCloneFrom object.
   - Clarified how and under which conditions the usmUserAuthProtocol
   and usmUserPrivProtocol can be initialized and/or changed.
   - Added comment on typical sizes for usmUserAuthKeyChange and
   usmUserPrivKeyChange. Also for usmUserOwnAuthKeyChange and
   usmUserOwnPrivKeyChange.
   - Added clarifications to the DESCRIPTION clauses of
   usmUserAuthKeyChange, usmUserOwnAuthKeychange, usmUserPrivKeyChange
   and usmUserOwnPrivKeychange. - Added clarification to DESCRIPTION
   clause of usmUserStorageType. - Added clarification to DESCRIPTION
   clause of usmUserStatus.
   - Clarified IV generation procedure in section 8.1.1.1 and in
   addition clarified section 8.3.1 step 1 and section 8.3.2. step 3.
   - Clarified section 11.2 and added a warning that different size
   passwords with repetitive strings may result in same key.
   - Added template users to appendix A for cloning process.
   - Fixed C-code examples in Appendix A.
   - Fixed examples of generated keys in Appendix A.
   - Added examples of KeyChange values to Appendix A.
   - Used PDU Classes instead of RFC1905 PDU types.
   - Added text in the security section about Reports and Access Control
   to the MIB
   - Removed a incorrect note at the end of section 3.2 step 7.
   - Added a note in section 3.2 step 3.
   - Corrected various spelling errors and typos.
   - Corrected procedure for 3.2 step 2.a)
   - various clarifications.
   - Fixed references to new/revised documents
   - Change to no longer cache data that is not used

C. Full Copyright Statement

   Copyright (C) The Internet Society (1999). All Rights Reserved.

   This document and translations of it may be copied and furnished to
   others, and derivative works that comment on or otherwise explain it
   or assist in its implementation may be prepared, copied, published
   and distributed, in whole or in part, without restriction of any
   kind, provided that the above copyright notice and this paragraph are
   included on all such copies and derivative works. However, this
   document itself may not be modified in any way, such as by removing
   the copyright notice or references to the Internet Society or other
   Internet organizations, except as needed for the purpose of
   developing Internet standards in which case the procedures for
   copyrights defined in the Internet Standards process must be
   followed, or as required to translate it into languages other than
   English.

   The limited permissions granted above are perpetual and will not be
   revoked by the Internet Society or its successors or assigns.

   This document and the information contained herein is provided on an
   "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING
   TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
   BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
   HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
   MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

   Funding for the RFC Editor function is currently provided by
   the Internet Society.