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

18 June 1997

U. Blumenthal
IBM T. J. Watson Research
uri@watson.ibm.com

B. Wijnen
IBM T. J. Watson Research
wijnen@vnet.ibm.com

<draft-ietf-snmpv3-usec-01.txt>

Status of this Memo

Abstract

This document describes the User-based Security Model (USEC) for SNMP version 3 for use in the SNMP architecture [SNMP-ARCH]. This document 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.

0.1 Issues

- Do we indeed want to move all STATS counters to MPC, we have assumed so for now.
- Do we need to do group mapping here and pass it back to MPC we have assumed so for now... but other documents do not pass groupName around.

- Do we want to check reportableFlag to determine if caching of securityData is needed or not.

0.2 Change Log

[version 1.2]
- changed (simplified) time sync in section 3 item 7.
- added usecUserMiId
- cleaned up text
- defined IV "salt" generation
- removed Statistics counters (now in MPC) and reportPDU generation (now in MPC)
- Removed auth and des MIBs which are now merged into USEC MIB
- specified where cachedSecurityData needs to be discarded
- added abtract service interface definitions
- removed section on error reporting (is MPC responsibility)
- removed auth/priv protocol definitions, they are in ARCH now
- removed MIB definitions for snmpEngineID,Time,Boots. They are in ARCH now.

[version 1.1]
- removed <securityCookie>.
- added <securityIdentity>, <securityCachedData>.
- added abstract function interface description of inter-module communications.
- modified IV generation process to accomodate messages produced faster than one-per-second (still open).
- always update the clock regardless of whether incoming message was Report or not (if the message was properly authenticated and its timestamp is ahead of our notion of their clock).

[version 1.0]
- first version posted to the v3editors mailing list.
- based on v2adv slides, v2adv items and issues list and on RFC1910 and SNMPv2u and SNMPv2* documents.
- various iterations were done by the authors via private email.

1. Introduction

The Architecture for describing Internet Management Frameworks is composed of multiple subsystems:

1) a message processing and control subsystem,
2) a security subsystem,
3) an access control subsystem, and
4) orangelets.

It is important to understand the SNMP architecture and the terminology of the architecture to understand where the 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 [SNMP-ARCH].

This memo [SNMPv3-USEC] 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) to associate security information with.

This memo describes the use of Keyed-MD5 as the authentication protocol 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. So the description of Keyed-MD5 and CBC-DES are in separate sections. That way it shows that they are supposed to be self-contained pieces that can be replaced or supplemented in the future.

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 SNMPv3 security model should provide protection are:

- Modification of Information
  The modification threat is the danger that some unauthorized entity may alter in-transit SNMPv3 messages generated on behalf of an authorized user 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 protocols defined in this memo provide 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 SNMPv3 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 SNMPv3 security protocol need not protect against. The security protocols defined in this memo do not provide protection against:

- Denial of Service
  An SNMPv3 security protocol need 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
  In addition, an SNMPv3 security protocol need not attempt to address traffic analysis attacks. Indeed, many traffic patterns are predictable - agents may be managed on a regular basis by a relatively small number of management stations - 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 SNMPv3 security model are as follows.

1) The protocol should provide for verification that each received SNMPv3 message has not been modified during its transmission through the network in such a way that an unauthorized management operation might result.

2) The protocol should provide for verification of the identity of

the user on whose behalf a received SNMPv3 message claims to have been generated.

3) The protocol should provide for detection of received SNMPv3 messages, which request or contain management information, whose time of generation was not recent.

4) The protocol should provide, when necessary, that the contents of each received SNMPv3 message are protected from disclosure.

In addition to the principal goal of supporting secure network management, the design of this SNMPv3 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 an SNMPv3 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.

For the protocols specified in this memo, it is not possible to assure the specific originator of a received SNMPv3 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. Implementation Organization

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

- The timeliness module must provide for:

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

- The authentication module must provide for:

  - Data Integrity,

  - Data Origin Authentication

- The privacy module must provide for

  - Protection against disclosure of the message payload.

The timeliness module is fixed for this 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 time values in an SNMPv3 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 time values in a message that passes the authentication module are trustworthy.

1.4.2. Authentication Protocol

Section 6 describes the Keyed-MD5 authentication protocol which is the first authentication protocol to be used with the User-based Security model. In the future additional or replacement

authentication protocols may be defined as new needs arise.

This User-based Security model prescribes that 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 7 describes the CBC-DES Symmetric Encryption Protocol which 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.

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

This 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 engine. For messages with a GET, GETNEXT, GETBULK, SET or INFORM request as the payload, the receiver of such messages is authoritative. For messages with a SNMPv2-TRAP, RESPONSE or REPORT as the payload, the sender is authoritative.

1.5.2 The following mechanisms are used:

- To protect against the threat of message delay or replay (to an extent greater than can occur through normal operation), a set of time (at the authoritative source) indicators and a request-id are included in each message generated. An SNMPv3 engine evaluates the time indicators to determine if a received message is recent. An SNMPv3 engine may evaluate the time 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 SNMPv3 engine uses received authentic messages to advance its notion of time at the remote authoritative source. An SNMPv3 engine also evaluates the request-id in received Response messages and discards messages which do not correspond to outstanding requests.

  These mechanisms provide for the detection of messages whose time of generation was not recent in all but one circumstance; this circumstance is the delay or replay of a Report message (sent to a

receiver) when the receiver has has not recently communicated with the source of the Report message. In this circumstance, the detection guarantees only that the Report message is more recent than the last communication between source and destination of the

Report message. However, Report messages do not request or contain management information, and thus, goal #3 in Section 1.2 above is met; further, Report messages can at most cause the receiver to advance its notion of time (at the source) by less than the proper amount.

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 SNMPv3 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]).

- verifying that a message sent to/from one SNMPv3 engine cannot be replayed to/as-if-from another SNMPv3 engine.

  Included in each message is an identifier unique to the SNMPv3 engine associated with the sender or intended recipient of the message. Also, each message containing a Response PDU contains a request-id which associates the message to a recently generated request.

  A Report message sent by one SNMPv3 engine to a second SNMPv3 engine can potentially be replayed to another SNMPv3 engine. However, Report messages do not request or contain management information, and thus, goal #3 in Section 1.2 above is met; further, Report messages can at most cause the receiver to advance its notion of time (at the authoritative source) by less than the correct amount.

- detecting messages which were not recently generated.

  A set of time indicators are included in the message, indicating the time of generation. Messages (other than those containing Report PDUs) without recent time indicators are not considered authentic. In addition, messages containing Response PDUs have a request-id; if the request-id does not match that of a recently generated request, then the message is not considered to be authentic.

  A Report message sent by an SNMPv3 engine can potentially be replayed at a later time to an SNMPv3 engine which has not recently communicated with that source engine. However, Report messages do not request or contain management information, and

thus, goal #3 in Section 1.2 above is met; further, Report messages can at most cause the receiver to advance its notion of time (at the authoritative source) by less than the correct amount.

This memo allows the same user to be defined on multiple SNMPv3 engines. Each SNMPv3 engine maintains a value, snmpEngineID, which uniquely identifies the engine. This value is included in each message sent to/from the engine that is authoritative (see section 1.5.1). On receipt of a message, an authoritative engine checks the value to ensure it is the intended recipient, and a non-authoritative engine uses the value to ensure that the message is processed using the correct state information.

Each SNMPv3 engine maintains two values, engineBoots and engineTime, which taken together provide an indication of time at that engine. Both of these values are included in an authenticated message sent to/received from that engine. On receipt, the values are checked to ensure that the indicated time 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 SNMPv3 engine to generate a message which an authoritative engine will accept as authentic, and to verify that a message received from that authoritative engine is authentic, such an engine must first achieve time synchronization with the authoritative engine.

2. Elements of the Model

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

2.1. SNMPv3 Users

Management operations using this security model make use of a defined set of user identities. For any SNMPv3 user on whose behalf management operations are authorized at a particular SNMPv3 engine, that engine must have knowledge of that user. An SNMPv3 engine that wishes to communicate with another SNMPv3 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.

<miId>
A human-readable string representing a (security) model independent identity for this user.

<groupName>
A string representing the group that the user belongs to.

<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. One such protocol is defined in this memo: the Digest 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 engines. Not visible via remote access.

<authKeyChange>
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 DES-based 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 engines. Not visible via remote access.

<privKeyChange>
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 SNMPv3 engine maintains three objects:

- snmpEngineID, which is an identifier unique among all SNMPv3 engines in (at least) an administrative domain;

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

- engineTime, which is the number of seconds since engineBoots was last incremented.

Each SNMPv3 engine is always authoritative with respect to these objects in its own engine. It is the responsibility of a non-authoritative SNMPv3 engine to synchronize with the authoritative engine, as appropriate.

An authoritative SNMPv3 engine is required to maintain the values of its snmpEngineID and engineBoots in non-volatile storage.

2.2.1. snmpEngineID

The engineID value contained in an authenticated message is used to defeat attacks in which messages from one engine to another engine are replayed to a different engine.

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

2.2.2. engineBoots and engineTime

The engineBoots and engineTime values contained in an authenticated message are used to defeat attacks in which messages are replayed

when they are no longer valid. Through use of engineBoots and engineTime, there is no requirement for an SNMPv3 engine to have a

non-volatile clock which ticks (i.e., increases with the passage of time) even when the engine is powered off. Rather, each time an SNMPv3 engine re-boots, it retrieves, increments, and then stores engineBoots in non-volatile storage, and resets engineTime to zero.

When an SNMPv3 engine is first installed, it sets its local values of engineBoots and engineTime to zero. If engineTime ever reaches its maximum value (2147483647), then engineBoots is incremented as if the engine has re-booted and engineTime is reset to zero and starts incrementing again.

Each time an authoritative SNMPv3 engine re-boots, any SNMPv3 engines holding that authoritative engine's values of engineBoots and engineTime need to re-synchronize prior to sending correctly authenticated messages to that authoritative 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 engine, when sent by an authoritative engine which has re-booted since the receiving engine last (re-)synchronized.

If an authoritative SNMPv3 engine is ever unable to determine its latest engineBoots value, then it must set its engineBoots value to 0xffffffff.

Whenever the local value of engineBoots has the value 0xffffffff, it latches at that value and an authenticated message always causes an notInTimeWindow authentication failure.

In order to reset an engine whose engineBoots value has reached the value 0xffffffff, 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 engine.

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 engine (see section 5.1.1) in order to proceed with authentic communications, has occurred when the non-authoritative engine has obtained local values of engineBoots and engineTime from the authoritative engine that are within the authoritative engine's time window. To remain synchronized, the local values must remain within the authoritative

engine's time window and thus must be kept loosely synchronized with the values stored at the authoritative engine.

In addition to keeping a local version of engineBoots and engineTime, a non-authoritative engine must also keep one other local variable, latestReceivedEngineTime. This value records the highest value of engineTime that was received by the non-authoritative engine from the authoritative engine and is used to eliminate the possibility of replaying messages that would prevent the non-authoritative engine's notion of the engineTime from advancing.

Time synchronization occurs as part of the procedures of receiving a message (Section 3.2, step 7b). As such, no explicit time synchronization procedure is required by a non-authoritative engine. Note, that whenever the local value of snmpEngineID is changed (e.g., through discovery) or when secure communications are first established with this engine, the local values of engineBoots and latestReceivedEngineTime should be set to zero. This will cause the time synchronization to occur when the next authentic message is received.

2.4. SNMPv3 Messages Using this Model

The syntax of an SNMPv3 message using this security model adheres to the message format defined in the SNMP Architecture document [SNMP-ARCH]. The securityParameters in the message are defined as an OCTET STRING. The format of that OCTET STRING for the User-based Security model is as follows:

```
securityParameters ::=
    SEQUENCE {
        -- global parameters
        engineID
            OCTET STRING (SIZE(12)),
        engineBoots
            Unsigned32 (0..4294967295),
        engineTime
            Unsigned32 (0..2147483647),
        userName
            OCTET STRING (SIZE(1..16)),
        authParameters
            OCTET STRING,
        privParameters
            OCTET STRING,
    }
END
```

The authParameters are defined by the authentication protocol in use for the message (as defined by the authProtocol column in the user's entry in the usecUserTable).

The privParameters are defined by the privacy protocol in use for the message (as defined by the privProtocol column in the user's entry in the usecUserTable).

2.5 Input and Output of the User-based Security Module

This section describes the inputs and outputs that the User-based Security module expects and produces when the Message Processing and Control module (MPC) invokes the User-base Security module for services.

2.5.1 Input and Output when generating an SNMPv3 Message

When the Message Processing and Control module (MPC) invokes the User-based Security module to secure an outgoing SNMPv3 message, there are two possibilities:

a) A new request is generated. The abstract service interface is:

```
generateRequestMsg(version, msgID, mms, msgFlags,
                   securityModel, securityParameters,
                   LoS, miId, engineID, scopedPDU)
```

b) A response is generated. The abtract service interface is:

```
generateResponseMsg(version, msgID, mms, msgFlags,
                    securityModel, securityParameters,
                    scopedPDU, cachedSecurityDataReference)
```

Where:

version
 This is the version number for the SNMP message.
 This data is not used by the USEC module.
 It is part of the globalData of the message.
msgID
 This is the msgID to be generated.
 This data is not used by the USEC module.
 It is part of the globalData of the message.
mms
 This is the maximum message size.
 This data is not used by the USEC module.
 It is part of the globalData of the message.
msgFlags
 This is the field containing the msgFlags.
 This data is not used by the USEC module.
 It is part of the globalData of the message.
 It should be consistent with the LoS that is passed.
securityModel
 This is the securityModel in use. Should be the USEC model.
 This data is not used by the USEC module.
 It is part of the globalData of the message.

securityParameters
 These are the security parameters. They will be filled in

by the User-based Security module.
LoS
The Level of Security (LoS) 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.
scopedPDU
this is the message payload. The data is opaque as far as the User-based Security module is concerned.
miId
this is the (security) model independent Identifier. Together with the engineID it identifies a row in the usecUserTable that is to be used for securing the message.
engineID
the engineID of the authoritative SNMP engine to which the request is to be sent.
cachedSecurityDataReference
A handle/reference to cached security data to be used when securing an outgoing response. This is the handle/reference that was generated by the USEC module when the incoming request was processed.

Upon completion of the process, the User-based Security module returns either and error indication or the completed message with privacy and authentication applied if such was requested by the Level of Security (LoS) flags passed.

The abstract service interface is:

returnGeneratedMsg(wholeMsg, wholeMsgLen, statusCode)

Where:
wholeMsg
this is fully encoded and secured message ready to be sent on the wire.
wholeMsgLen
this is the length of the encoded and secured message wholeMsg.
statusCode
this is the indicator of whether the encoding and securing of the message was successful, and if not it is an indication of the problem.

2.5.2 Input and Output when receiving an SNMPv3 Message

The Message Processing and Control module (MPC) invokes the User-based Security module to verify proper security of an incoming SNMPv3 message. The abstract service interface is:

processMsg(version, msgID, mms, msgFlags,

```
                securityModel, securityParameters,
                LoS, wholeMsg, wholeMsgLen)
```

Where:

version
This is the version number for the SNMP message.
This data is not used by the USEC module.
It is part of the globalData of the message.
msgID
This is the msgID to be generated.
This data is not used by the USEC module.
It is part of the globalData of the message.
mms
This is the maximum message size.
This data is not used by the USEC module.
It is part of the globalData of the message.
msgFlags
This is the field containing the msgFlags.
This data is not used by the USEC module.
It is part of the globalData of the message.
It should be consistent with the LoS that is passed.
securityModel
This is the securityModel in use. Should be the USEC model.
This data is not used by the USEC module.
It is part of the globalData of the message.
securityParameters
These are the security parameters. They will be filled in
by the User-based Security module.
LoS
The Level of Security (LoS) 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
this is the complete message as it was received by the Message
Processing and Control module (MPC).
wholeMsgLen
this is the length of the wholeMsg as received on the wire.

Upon completion of the process, the User-based Security module returns a statusCode and in case of success authenticated and decrypted data. The abstract service interface is:

```
returnMsg(miId, groupName, cachedSecurityDataReference,
          scopedPDUmms, scopedPDU, statusCode)
```

Where:

miId
this is an Security Model-independent Identifier that identifies
an entry in the usecUserTable. It is to be used later when a

response message must be secured.

groupName

this is the group to which the user belongs. The User-based

Security module retrieves this information from the usecUserTable.

cachedSecurityDataReference
cached security data to be used when securing a possible outgoing response to this request. Will have to be released explicitly by the MPC or the application.

scopedPDUmms
this is the maximum message size that a possible response PDU may use. The User-based Security module calculates this size such that there is always space available for any security parameters that need to be added to the response message.

scopedPDU
this is the message payload. The data is opaque as far as the User-based Security module is concerned. But if the data was encrypted because privacy protection was in effect, then upon return from the User-based Security module the data will have been decrypted.

statusCode
this is an indicator of whether the message was parsed, authenticated and possibly decrypted successfully. If it was not - it indicates what the problem was.

3. Elements of Procedure

This section describes the security related procedures followed by an SNMPv3 engine when processing SNMPv3 messages according to the User-based Security model.

3.1. Processing an Outgoing Message

This section describes the procedure followed by an SNMPv3 engine whenever it generates a message containing a management operation (either a request, a response, a notification, or a report) on behalf of a user, with a particular Level of Security (LoS).

1) - If any cachedSecurityDataReference is passed, then information concerning the user is extracted from the cachedSecurityData. The cachedSecurityData can now be discarded.
   - Otherwise, based on the miId, information concerning the user at the destination engineID is extracted from the Local (security) Configuration Datastore (LCD, usecUserTable). If information about the user is absent from the LCD, then an error indication (unknownSecurityIdentity) is returned to the calling module.

2) If the Level of Security (LoS) 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 (unsupportedLoS) is returned to the calling module.

3) If the Level of Security (LoS) 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 (unsupportedLoS) is returned to the calling module.

4) If the Level of Security (LoS) 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. The abstract service interface is:

   encryptMsg(cryptKey, scopedPDU)

   Where:

   cryptKey
   The user's usecUserPrivKey. This is the secret key that can be used by the encryption algorithm.

scopedPDU
  The data to be encrypted.

Upon completion the privacy module returns:

returnEncryptedMsg(encryptedPDU, privParameters, statusCode)

encryptedPDU
The encrypted scopedPDU (encoded as an octet string).
privParameters
The privacy parameters (encoded as an octet string) that need to be sent in the outgoing message.
statusCode
The indicator of whether the PDU was encrypted successfully and if not, it indicates what went wrong.

If an error indication is returned by the privacy module then the message cannot be sent and the error indication is returned to the calling module.

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

5) If the Level of Security (LoS) specifies that the message is not to be protected from disclosure, then the NULL string is encoded as an octet string into the privParameters field of the securityParameters and the scopedPDU serves as the payload of the message being prepared.

6) The engineID is encoded as an octet string into the <engineID> field of the securityParameters.

7) If the Level of Security (LoS) specifies that the message is to be authenticated, then the current values of engineBoots, and engineTime corresponding to engineID from the LCD are used. Otherwise, a zero value is used for engineBoots and engineTime. The values are encoded as Unsigned32 into the engineBoots and engineTime fields of the securityParameters.

8) The userName is encoded as an octet string into the userName field of the securityParameters.

9) If the Level of Security (LoS) 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. The abstract service interface is:

authMsg(authKey, wholeMsg)

authKey
  The user's usecUserAuthKey. This is the secret key

that can be used by the authentication algorithm.
wholeMsg
the message to be authenticated.

Upon completion the authentication module returns:

returnAuthMsg(wholeMsg, statusCode)

wholeMsg
Same as in input, but with authParameters properly filled.
statusCode
The indicator of whether the message was successfully processed by the authentication module.

If an error indication is returned by the authentication module, then the message cannot be sent and the error indication is returned to the calling module.

10) If the Level of Security (LoS) specifies that the message is not to be authenticated then the NULL string is encoded as an octet string into the authParameters field of the securityParameters.

11) The completed message is returned to the calling module with the statusCode set to success.

3.2. Processing an Incoming Message

This section describes the procedure followed by an SNMPv3 engine whenever it receives a message containing a management operation on behalf of a user, with a particular Level of Security (LoS).

1) If the received securityParameters is not the serialization (according to the conventions of [RFC1906]) of an OCTET STRING formated according to the securityParameters defined in section 2.4, then the snmpInASNParseErrs counter [RFC1907] is incremented, and an error indication (ASNParseError) is returned to the calling module.

2) The values of the security parameter fields are extracted from the securityParameters.

3) If the engineID field contained in the securityParameters is unknown then:

- a manager that performs discovery may optionally create a new entry in its Local (security) Configuration Database (LCD) and continue processing; or

- an error indication (unknownEngineID) is returned to the

calling module.

4) Information about the value of the userName and engineID fields is extracted from the Local (security) Configuration Database (LCD, usecUserTable). If no information is available for this user, then an error indication (unknownSecurityIdentity) is returned to the calling module.

5) If the information about the user indicates that it does not support the Level of Security indicated by the LoS parameter, then and an error indication (unsupportedLoS) is returned to the calling module.

6) If the Level of Security (LoS) 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. The abstract service interface is:

```
    authIncomingMsg(authKey, authParameters, wholeMsg)
```

```
  authKey
    The user's (secret) usecUserAuthKey
  authParameters
    the authParameters from the incoming message.
  wholeMsg
    the message to be authenticated.
```

The authentication module returns:

```
    returnAuthIncomingMsg(wholeMsg, statusCode)
```

If the message is not authentic according to the authentication protocol module (i.e. it returns an error indication), then the error indication is returned to the calling module.

Otherwise, the authenticated wholeMsg is used for further processing.

7) If the LoS field indicates an authenticated message, then the local values of engineBoots and engineTime corresponding to the value of the engineID field are extracted from the Local (security) Configuration Database (LCD).

a) If the engineID value is the same as the snmpEngineID of the processing SNMPv3 engine (meaning that this is the authoritative 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 engineBoots is 0xffffffff;

- the engineBoots field differs from the local value of

engineBoots; or,

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

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

b) If the engineID value is not the same as the snmpEngineID of the processing SNMPv3 engine (meaning that this engine is not the authoritative engine), then:

- if at least one of the following conditions is true:

  - the engineBoots field is greater than the local value of engineBoots; or,

  - the engineBoots field is equal to the local value of engineBoots and the engineTime field is greater than the value of latestReceivedEngineTime,

  then the LCD entry corresponding to the value of the engineID field is updated, by setting the local value of engineBoots from the engineBoots field, the local value latestReceivedEngineTime from the engineTime field, and the local value of engineTime from the engineTime field.

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

  - the local value of engineBoots is 0xffffffff;

  - the engineBoots field is less than the local value of engineBoots; or,

  - the engineBoots field is equal to the local value of engineBoots and the engineTime field is more than 150 seconds less than the local notion of engineTime.

  If the message is considered to be outside of the Time Window then an error indication (notInTimeWindow) is returned to the calling module;
  however, time synchronization procedures may be invoked.
  Note that this procedure allows for engineBoots in the message to be greater than the local value of engineBoots to allow for received messages to be accepted as authentic when received from an authoritative SNMPv3 engine that has re-booted since the receiving SNMPv3 engine last

(re-)synchronized.

8) If the LoS field indicates that the message was protected from disclosure, then the octet sequence representing the scopedPDU is decrypted according to the user's privacy protocol to obtain a serialized scopedPDUs value. Otherwise the data component is assumed to directly contain the scopedPDUs value. To do the decryption, a call is made to the privacy module that implements the user's privacy protocol. The abstract service interface is:

```
    decryptMsg(cryptKey, privParameters, encryptedPDU)

  cryptKey
    The user's secret usecUserPrivKey
  privParameters
    The privParameters field from the securityParameters from
    the incoming message.
  encryptedPDU
    the data to be decrypted
```

The privacy module returns:

```
    returnDecryptedMsg(scopedPDU, statusCode)

  scopedPDU
    The decrypted scopedPDU.
  statusCode
    The indicator whether the message was successfully decrypted.
```

If an error indication is returned by the privacy module, then the error indication is returned to the calling module.

9) The scopedPDU-MMS is calculated.

10) The groupName is retrieved from the usecUserTable

11) The miId is retrieved from the usecUserTable

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

```
   usecUserName,
   usecUserAuthProto, usecUserAuthKey,
   usecUserPrivProto, usecUserPrivKey
```

-- Editor's note:
   If we assume SNMPv3, then we could check the reportableFlag and if it is not set, then we do not need to cache any security data because then there is no response possible. Do we want to do that?
-- End Editor's note.

13) The statusCode is set to success and a return is made to the

calling module according to this abstract service interface:

```
returnMsg(miId, groupName, cachedSecurityDataReference,
          scopedPDUmms, scopedPDU, statusCode)
```

4. Discovery

This security model requires that a discovery process obtains sufficient information about other SNMP engines in order to communicate with them. Discovery requires the SNMP manager to learn the engine's snmpEngineID value before communication may proceed. This may be accomplished by formulating a get-request communication with the LoS set to noAuth/noPriv, the userName set to "public", the snmpEngineID set to all zeros (binary), and the varBindList left empty. The response to this message will be a report PDU that contains the snmpEngineID within the securityParameters field (and containing the snmpUnknownEngineIDs counter in the varBindList).
If authenticated communication is required then the discovery process may invoke the procedure described in Section 2.3 to synchronize the timers.

5. Definitions

```
SNMP-USEC-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, snmpModules  FROM SNMPv2-SMI
    TEXTUAL-CONVENTION, TestAndIncr,
    RowStatus, StorageType                     FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP            FROM SNMPv2-CONF,
    SnmpAdminString, SnmpLoS, SnmpEngineID,
    SnmpSecurityModel,
    imfAuthMD5Protocol, imfNoPrivProtocol      FROM IMF-MIB;

snmpUsecMIB MODULE-IDENTITY
    LAST-UPDATED "9706180000Z"     -- 18 June 1997, midnight
    ORGANIZATION "SNMPv3 Working Group"
    CONTACT-INFO "WG-email:   snmpv3@tis.com
                  Subscribe:  majordomo@tis.com
                              In msg body:  subscribe snmpv3

                  Chair:      Russ Mundy
                              Trusted Information Systems
                  postal:     3060 Washington Rd
                              Glenwood MD 21738
                  email:      mundy@tis.com
                  phone:      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
                  SNMPv3 User-based Security model.
                 "
    ::= { snmpModules 99 }  -- to be assigned
```

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

```
snmpUsecAdmin           OBJECT IDENTIFIER ::= { snmpUsecMIB 1 }
snmpUsecMIBObjects      OBJECT IDENTIFIER ::= { snmpUsecMIB 2 }
snmpUsecMIBConformance  OBJECT IDENTIFIER ::= { snmpUsecMIB 3 }

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

UserName ::=     TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION "A string representing the name of a user for use in
                 accordance with the SNMP User-based Security model.
                "
    SYNTAX       SnmpAdminString (SIZE(1..16))


-- Editor's note:
-- A real issue is whether the fact that MD5 is used in the following
-- TC is OK. It might be better to use 3DES for 3DES and IDEA for IDEA.
-- End Editor's note

KeyChange ::=    TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
          "Every definition of an object with this syntax must identify
           a protocol, P, and a secret key, K.  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: a 'random' component and 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,
           imfAuthMD5Protocol requires K to be a fixed length of 16
           octets.  Other protocols may define other sizes, as deemed
           appropriate.

           When an instance of this object is modified to have a new
           value by the management protocol, the agent generates a new
           value of K as follows:

            - a temporary variable is initialized to the existing value
              of K;
            - if the length of the delta component is greater than 16
```

bytes, then:
 - the random component is appended to the value of the

```
                temporary variable, and the result is input to the MD5
                hash algorithm 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) 16-bytes of the delta component to produce
                the first (next) 16-bytes of the new value of K.
              - the above two steps are repeated until the unused
                portion of the delta component is 16 bytes or less,
            - the random component is appended to the value of the
              temporary variable, and the result is input to the MD5
              hash algorithm 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.

              i.e.,

                 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."
    SYNTAX      OCTET STRING

-- ************************************************************

-- The valid users for the User-based Security model *****************

usecUser         OBJECT IDENTIFIER ::= { snmpUsecMIBObjects 1 }

usecUserTable    OBJECT-TYPE
    SYNTAX       SEQUENCE OF UsecUserEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The table of users configured in the SNMP engine's
                 Local (security) Configuration Datastore (LCD)."
    ::= { usecUser 1 }

usecUserEntry    OBJECT-TYPE
```

```
    SYNTAX       UsecUserEntry
    MAX-ACCESS   not-accessible
```

```
    STATUS       current
    DESCRIPTION "A user configured in the SNMP engine's Local
                 (security) Configuration Datastore (LCD) for
                 the User-based Security model.
                "
    INDEX       { usecUserEngineID,
                  IMPLIED usecUserName
                }
    ::= { usecUserTable 1 }

UsecUserEntry ::= SEQUENCE {
    usecUserEngineID         SnmpEngineID,
    usecUserName             UserName,
    usecUserMiId             SnmpAdminString,
    usecUserGroupName        SnmpAdminString,
    usecUserCloneFrom        RowPointer,
    usecUserAuthProtocol     OBJECT IDENTIFIER,
    usecUserAuthKeyChange    KeyChange,
--  usecUserAuthKey          OCTET STRING, not visible
    usecUserAuthPublic       OCTET STRING,
    usecUserPrivProtocol     OBJECT IDENTIFIER,
    usecUserPrivKeyChange    KeyChange,
--  usecUserPrivKey          OCTET STRING, not visible
    usecUserPrivPublic       OCTET STRING,
    usecUserStorageType      StorageType,
    usecUserStatus           RowStatus
}

usecUserEngineID 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.

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

usecUserName     OBJECT-TYPE
    SYNTAX       UserName
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "A string representing the name of the user.  This is
```

```
                the (User-based security) model dependent identity.
               "
    ::= { usecUserEntry 2 }
```

```
usecUserMiId     OBJECT-TYPE
    SYNTAX       SnmpAdminString
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "A string representing the (security) model independent
                 identity for this user.

                 The default mapping for the User-based Security model
                 is that the miId is the same as the userName.
                "
    ::= { usecUserEntry 3 }

usecUserGroupName OBJECT-TYPE
    SYNTAX       SnmpAdminString
    MAX-ACCESS   read-write
    STATUS       current
    DESCRIPTION "A string representing the group to which the user
                 belongs.  A group name of zero length indicates
                 that the user is not [perhaps yet] a member of any
                 group, possibly because the entry has not yet been
                 completely configured.  Users which are not a part
                 of any group are effectively disabled to perform any
                 SNMP operations.
                "
    DEFVAL      { ''H } -- the empty string
    ::= { usecUserEntry 4 }

usecUserCloneFrom OBJECT-TYPE
    SYNTAX       RowPointer
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "A pointer to another conceptual row in this
                 usecUserTable.  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 authentication
                 parameters of the new user are cloned from its
                 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 agent.
                 The cloning process fails with an 'inconsistentName'
                 error if the conceptual row representing the
```

clone-from user is not in an active state when the
cloning process is invoked.

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

                When this object is read, the zero length string is
                returned.
               "
    ::= { usecUserEntry 5 }

usecUserAuthProtocol OBJECT-TYPE
    SYNTAX       OBJECT IDENTIFIER
    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
                usecUserEngineID, 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).  Once created,
                the value of an instance of this object can not be
                changed.
               "
    DEFVAL      { imfAuthMD5Protocol }
    ::= { usecUserEntry 6 }

usecUserAuthKeyChange OBJECT-TYPE
    SYNTAX       KeyChange  -- typically (SIZE (0..32))
    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
                usecUserEngineID, to be modified via a one-way
                function.

                The associated protocol is the usecUserAuthProtocol.
                The associated secret key is the user's secret
                authentication key (usecUserAuthKey).

                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 value
```

```
                of usecUserCloneFrom.
               "
    DEFVAL      { ''H }    -- the empty string
```

```
    ::= { usecUserEntry 7 }

usecUserAuthPublic    OBJECT-TYPE
    SYNTAX       OCTET STRING -- for MD5 (SIZE(0..32))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "A publicly-readable value which is written as part
                 of the procedure for changing a user's secret key,
                 and later read to determine whether the change of
                 the secrets was effected.
                "
    DEFVAL      { ''H }  -- the empty string
    ::= { usecUserEntry 8 }

usecUserPrivProtocol OBJECT-TYPE
    SYNTAX       OBJECT IDENTIFIER
    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
                 usecUserEngineID, 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).  Once created,
                 the value of an instance of this object can not be
                 changed.
                "
    DEFVAL      { imfNoPrivProtocol }
    ::= { usecUserEntry 9 }

usecUserPrivKeyChange OBJECT-TYPE
    SYNTAX       KeyChange  -- typically (SIZE (0..32))
    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
                 usecUserEngineID, to be modified via a one-way
                 function.

                 The associated protocol is the usecUserPrivProtocol.
                 The associated secret key is the user's secret
                 encryption key (usecUserPrivKey).
```

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 value
                of usecUserCloneFrom.
               "
    DEFVAL     { ''H }    -- the empty string
    ::= { usecUserEntry 10 }

usecUserPrivPublic    OBJECT-TYPE
    SYNTAX       OCTET STRING -- for DES (SIZE(0..16))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "A publicly-readable value which is written as part
                of the procedure for changing a user's secret key,
                and later read to determine whether the change of
                the secrets was effected.
               "
    DEFVAL     { ''H }  -- the empty string
    ::= { usecUserEntry 11 }

usecUserStorageType OBJECT-TYPE
    SYNTAX       StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The storage type for this conceptual row."
    DEFVAL     { nonVolatile }
    ::= { usecUserEntry 12 }

usecUserStatus 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 usecUserStatus column is 'notReady'.

                For those columnar objects which permit write-access,
                their value in an existing conceptual row can be
                changed irrespective of the value of usecUserStatus
                for that row.
               "
    ::= { usecUserEntry 13 }


usecUserSecretSpinLock  OBJECT-TYPE
    SYNTAX       TestAndIncr
    MAX-ACCESS   read-write
    STATUS       current
    DESCRIPTION "An advisory lock used to allow several cooperating
```

SNMPv3 engines, all acting in a manager role, to coordinate their use of facilities to alter secrets in the usecUserTable.

```
                "
    ::= { usecUser 2 }

--Editor's note
Is it enough to have just one spin-lock for such a table where
several secrets can be modified? Can the protocol ensure the
consistency? Should it?
--End editor's note

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

snmpUsecMIBCompliances
                 OBJECT IDENTIFIER ::= { snmpUsecMIBConformance 1 }
snmpUsecMIBGroups
                 OBJECT IDENTIFIER ::= { snmpUsecMIBConformance 2 }

-- Compliance statements

snmpUsecMIBCompliance MODULE-COMPLIANCE
    STATUS       current
    DESCRIPTION "The compliance statement for SNMP engines which
                 implement the SNMP USEC MIB.
                "

    MODULE       -- this module
        MANDATORY-GROUPS { snmpUsecMIBBasicGroup }

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

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

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

    ::= { snmpUsecMIBCompliances 1 }

-- Units of compliance

snmpUsecMIBBasicGroup OBJECT-GROUP
    OBJECTS     {
                  usecUserMiId,
                  usecUserGroupName,
                  usecUserCloneFrom,
                  usecUserAuthProtocol,
```

```
                usecUserAuthKeyChange,
                usecUserAuthPublic,
```

```
                  usecUserPrivProtocol,
                  usecUserPrivKeyChange,
                  usecUserPrivPublic,
                  usecUserStorageType,
                  usecUserStatus
                }
    STATUS       current
    DESCRIPTION "A collection of objects providing for configuration
                 of an SNMP engine which implements the SNMP
                 User-based Security model.
                "
    ::= { snmpUsecMIBGroups 1 }

END
```

6. MD5 Authentication Protocol

This section describes the Keyed-MD5 authentication protocol.
This protocol is the first authentication protocol defined for
the User-based Security model.
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 SNMPv3 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 both inserted into, and appended to, the SNMPv3 message prior to computing the digest; the inserted value is overwritten prior to transmission, and the appended value is not transmitted. The secret value is shared by all SNMPv3 engines authorized to originate messages on behalf of the appropriate user.

- In order to not expose the shared secrets (keys) at all SNMPv3 engines in case one of the engines is compromised, such secrets (keys) are localized for each authoritative SNMPv3 engine, see [Localized-Key].

6.1.1. Digest Authentication Protocol

The Digest Authentication Protocol defined in this memo provides for:

- verifying 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 a message. The digest is computed by the originator of the message, transmitted with the message, and verified by the recipient of the message.

- verifying the user on whose behalf the message was generated.

  A secret value known only to SNMPv3 engines authorized to generate messages on behalf of a user is both inserted into, and appended to, the message prior to the digest computation. Thus, the verification of the user is implicit with the verification of the digest. (Note that the use of two copies of the secret, one near the start and one at the end, is recommended by [KEYED-MD5].)

This protocol uses the MD5 [MD5] message digest algorithm. A 128-bit digest is calculated over the designated portion of an SNMPv3 message

and included as part of the message sent to the recipient. The size of both the digest carried in a message and the private authentication key (the secret) is 16 octets.

6.2 Elements of the Digest Authentication Protocol

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

6.2.1. SNMPv3 Users

Authentication using this Digest Authentication protocol makes use of a defined set of user identities. For any SNMPv3 user on whose behalf a message must be authenticated at a particular SNMPv3 engine, that engine must have knowledge of that user. An SNMPv3 engine that wishes to communicate with another SNMPv3 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.

6.2.2. EngineID

The engineID value contained in an authenticated message specifies the authoritative SNMPv3 engine for that particular message. (see the definition of engineID in the SNMP Architecture document [SNMP-ARCH]).

The user's (private) authentication key is normally different at each authoritative SNMPv3 engine and so the snmpEngineID is used to select the proper key for the authentication process.

6.2.3. SNMPv3 Messages Using this Authentication Protocol

Messages using this authentication protocol carry an authParameters field as part of the securityParameters. For this protocol, the authParameters field is the serialized octet string representing the MD5 digest of 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 Input and Output of the MD5 Authentication Module

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

6.2.4.1 Input and Output when generating an SNMPv3 Message

This MD5 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. The abstract service interface is:

```
authMsg(authKey, wholeMsg)
```

Where:

authKey
  The secret key to be used by the authentication algorithm.
wholeMsg
  the message to be authenticated.

Upon completion the authentication module returns information. The abstract service interface is:

```
returnAuthMsg(wholeMsg, statusCode)
```

Where:

wholeMsg
  Same as in input, but with authParameters properly filled.
statusCode
  The indicator of whether the message was successfully processed or not.

Note, that <authParameters> is filled by the authentication module and this field should be already present in the <wholeMsg> before the MAC is generated.

6.2.4.2 Input and Output when receiving an SNMPv3 Message

This MD5 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. The abstract service interface is:

```
authIncomingMsg(authKey, authParameters, wholeMsg)
```

Where:

authKey
  The secret key to be used by the authentication algorithm.

authParameters
  the authParameters from the incoming message.

wholeMsg
the message to be authenticated.

Upon completion the authentication module returns information. The abstract service interface is:

returnAuthIncomingMsg(wholeMsg, statusCode)

wholeMsg
Same as in input, data has been authenticated.
statusCode
The indicator of whether the message was successfully processed or not.

6.3 Elements of Procedure

This section describes the procedures for the Keyed-MD5 authentication protocol.

6.3.1 Processing an Outgoing Message

This section describes the procedure followed by an SNMPv3 engine whenever it must authenticate an outgoing message using the imfAuthMD5Protocol.

1) The authParameters field is set to the serialization according to the rules in [RFC1906] of an octet string representing the secret (localized) authKey.

2) The secret (localized) authKey is then appended to the end of the wholeMsg.

3) The MD5-Digest is calculated according to [MD5]. Then the authParameters field is replaced with the calculated digest.

4) The wholeMsg (excluding the appended secret key) is then returned to the caller together with a statusCode of success.

6.3.2 Processing an Incoming Message

This section describes the procedure followed by an SNMPv3 engine whenever it must authenticate an incoming message using the imfAuthMD5Protocol.

1) If the digest received in the authParameters field is not 16 octets long, then an error indication (authenticationError) is returned to the calling module.

2) The digest received in the authParameters field is saved.

3) The digest in the authParameters field is replaced by the

secret (localized) authKey.

4) The secret (localized) authKey is then appended to the end of the wholeMsg.

5) The MD5-Digest is calculated according to [MD5].
The authParameters field is replaced with the digest value that was saved in step 2).

6) Then the newly calculated digest is compared with the digest saved in step 2). If the digests do not match, then an error indication (authenticationError) is returned to the calling module.

7) The wholeMsg (excluding the appended secret key) and a statusCode of success are then returned to the caller.

7. DES Privacy Protocol

This section describes the DES privacy protocol.
This protocol is the first privacy protocol defined for the
User-based Security model.
Over time, other privacy protocols may be defined either
as a replacement of this protocol or in addition to this protocol.

7.1 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 is in combination with a time value is used to create the en/decryption key and the initialization vector. The secret value is shared by all SNMPv3 engines authorized to originate messages on behalf of the appropriate user.

- In order to not expose the shared secrets (keys) at all SNMPv3 engines in case one of the engines is compromised, such secrets (keys) are localized for each authoritative SNMPv3 engine, see [Localized-Key].

7.1.1. Symmetric Encryption Protocol

The Symmetric Encryption Protocol defined in this memo provides support for data confidentiality. The designated portion of an SNMPv3 message is encrypted and included as part of the message sent to the recipient.

This memo requires that if data confidentiality is supported by an SNMPv3 engine, this engine must implement at least the Data Encryption Standard (DES) in the Cipher Block Chaining mode of operation.

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 (see [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.

7.1.1.1 DES key and Initialization Vector.

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

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

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

In order to ensure that IV for two different packets encrypted by the same key, are not the same (i.e. IV does not repeat) we need to "salt" the pre-IV with something unique per packet.
An 8-byte octet string is used as the "salt". The concatenation of the generating 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 a random value at boot time. The 32-bit snmpEngineBoots is converted to the first 4 bytes (Most Significant Byte first) of our "salt". The 32-bit integer is then converted to the last 4 bytes (Most Significant Byte first) of our "salt". The resulting "salt" is then XOR-ed with the pre-IV. The 8-byte salt is then put into the privParameters field as an octet-string. The "salt" integer is incremented by one and wraps when it reaches the maximum value.

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

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.

7.1.1.2 Data Encryption.

The data to be encrypted is treated as sequence of octets. Its

length should be an integral multiple of 8 - and if 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.

7.1.1.3 Data Decryption

Before decryption, the encrypted data length is verified.
If the length of the octet sequence to be decrypted is not an integral multiple of 8 octets, the processing of the octet sequence 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.

7.2 Elements of the DES Privacy Protocol

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

7.2.1. SNMPv3 Users

Data En/Decryption using this Symmetric Encryption Protocol makes use of a defined set of user identities. For any SNMPv3 user on whose behalf a message must be en/decrypted at a particular SNMPv3 engine, that engine must have knowledge of that user. An SNMPv3 engine that wishes to communicate with another SNMPv3 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>
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.

7.2.2. EngineID

The engineID value contained in an authenticated message specifies the authoritative SNMPv3 engine for that particular message. (see the definition of engineID in the SNMP Architecture document [SNMP-ARCH]).

The user's (private) privacy key is normally different at each authoritative SNMPv3 engine and so the snmpEngineID is used to select the proper key for the authentication process.

7.2.3. SNMPv3 Messages Using this Privacy Protocol

Messages using this privacy protocol carry a privParameters field as part of the securityParameters. For this protocol, the privParameters field is the serialized octet string representing the "salt" that was used to create the IV.

7.2.4 Input and Output of 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.

7.2.4.1 Input and Output when generating an SNMPv3 Message

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. The abstract service interface is:

```
encryptMsg(cryptKey, scopedPDU)
```

Where:

cryptKey
The secret key to be used by the encryption algorithm.
scopedPDU
The data to be encrypted.

Upon completion the privacy module returns information. The abstract service interface is:

```
returnEncryptedMsg(encryptedPDU, privParameters, statusCode)
```

Where:

encryptedPDU
The encrypted scopedPDU (encoded as an octet string).

privParameters
  The privacy parameters (encoded as an octet string) that

need to be sent in the outgoing message.
statusCode
The indicator of whether the PDU was encrypted successfully and if not, it indicates what went wrong.

7.2.4.2 Input and Output when receiving an SNMPv3 Message

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. The abstract service interface is:

decryptMsg(cryptKey, privParameters, encryptedPDU)

Where:

cryptKey
The secret key to be used by the decryption algorithm.
privParameters
The "salt" to be used to calculate the IV.
encryptedPDU
the data to be decrypted

Upon completion the privacy module returns information. The abstract service interface is:

returnDecryptedMsg(scopedPDU, statusCode)

Where:

scopedPDU
The decrypted scopedPDU.
statusCode
The indicator whether the message was successfully decrypted.

7.3 Elements of Procedure.

This section describes the procedures for the DES privacy protocol.

7.3.1 Processing an Outgoing Message

This section describes the procedure followed by an SNMPv3 engine whenever it must encrypt part of an outgoing message using the imfPrivDESProtocol.

1) The secret (localized) cryptKey are used to construct the DES encryption key, the "salt" and the DES pre-IV (as described in 7.1.1.1).

2) The authParameters field is set to the serialization according

to the rules in [RFC1906] of an octet string representing the
the "salt" string.

2) The scopedPDU is encrypted (as described in 7.1.1.2) and the encrypted data is serialized according to the rules in [RFC1906] as an octet string.

3) The the serialized octet string representing the encrypted scopedPDU together with the privParameters and a statusCode of success is returned to the caller.

7.3.2 Processing an Incoming Message

This section describes the procedure followed by an SNMPv3 engine whenever it must decrypt part of an incoming message using the imfPrivDESProtocol.

1) If the privParameters field is not an 8-byte octet string, then an error indication (privacyError) is returned to the calling module.

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

3) The secret (localized) cryptKey and the "salt" are then used to construct the DES decryption key and pre-IV (as described in 7.1.1.1).

4) The encryptedPDU is decrypted (as described in 7.1.1.3).

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

6) The decrypted scopedPDU and a statusCode of success are returned to the calling module.

8. Editor's Addresses

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:	+1-914-784-7064

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

9. Acknowledgements

This document is based on the recommendations of the SNMP Security and Administrative Framework Evolution team, comprised of

David Harrington (Cabletron Systems Inc.)
Jeff Johnson (Cisco)
David Levi (SNMP Research Inc.)
John Linn (Openvision)
Russ Mundy (Trusted Information Systems) chair
Shawn Routhier (Epilogue)
Glenn Waters (Nortel)
Bert Wijnen (IBM T. J. Watson Research)

Further a lot of "cut and paste" material comes from RFC1910 and from earlier draft documents from the SNMPv2u and SNMPv2* series.

Further more a special thanks is due to the SNMPv3 WG, specifically:
....

10. Security Considerations

10.1. Recommended Practices

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

- A management station must discard SNMPv3 responses for which neither the msgID nor the request-id component or the represented management information corresponds to any currently outstanding request.

  Although it would be typical for a management station 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).

- A management station must generate unpredictable msgIDs and request-ids in authenticated messages in order to protect against the possibility of message duplication (malicious or otherwise). For example, start operations with msgID and/or request-id 0 is not a good idea. Initializing them with a pseudorandom number and then incrementing by one would be acceptable.

- A management station 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 agent, a management station should delay sending successive messages to the managed agent until a positive acknowledgement is received for the previous message or until the previous message expires.

  No message ordering is imposed by the SNMPv3. 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 agent, 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, a management station 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 SNMPv2 set operations in order to provide a mechanism to ensure the processing of SNMPv2 messages occurs in a specific order.

- The frequency with which the secrets of an SNMPv3 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 the communication paths, more caution is prudent.

10.2 Defining Users

The mechanisms defined in this document employ the notion of "users" which map into "groups" and such "groups" have access rights. 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 a manager station application or set of manager station 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 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 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.

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 engine, but security requirements strongly discourage having the same key for more than one engine, SNMPv3 employs a compromise proposed in [Localized-key].
It derives the user keys for the SNMPv3 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 SNMPv3 engine from any combination of user's keys on SNMPv3 engines.

Note however, that if user's password is disclosed, key localization will not help and network security may be compromised in this case.

10.3. Conformance

To be termed a "Secure SNMPv3 implementation" based on the User-base Security model, an SNMPv3 implementation:

- must implement one or more Authentication Protocol(s). The MD5 Authentication Protocol defined in this memo is one such protocol.

- must, to the maximum extent possible, prohibit access to the secret(s) of each user about which it maintains information in a Local (security) Configuration Database (LCD) under all circumstances except as required to generate and/or validate SNMPv3 messages with respect to that user.

- must implement the SNMP USEC MIB.

In addition, an authoritative SNMPv3 engine must provide initial configuration in accordance with Appendix A.1.

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

11. References

[RFC1902] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S., Waldbusser, "Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[RFC1905] The SNMPv2 Working Group, 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] The SNMPv2 Working Group, 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] The SNMPv2 Working Group, 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.

[RFC1908] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework", RFC 1908, January 1996.

[SNMP-ARCH] The SNMPv3 Working Group, Harrington, D., Wijnen, B., "An Architecture for describing Internet Management Frameworks", draft-ietf-snmpv3-next-gen-arch-02.txt, June 1997.

[SNMPv3-MPC] The SNMPv3 Working Group, Wijnen, B., Harrington, D., "Message Processing and Control Model for version 3 of the Simple Network Management Protocol (SNMPv3)", draft-ietf-snmpv3-mpc-01.txt, June 1997.

[SNMPv3-ACM] The SNMPv3 Working Group, Wijnen, B., Harrington, D., "Access Control Model for Version 3 of the Simple Network Management Protocol (SNMPv3)", draft-ietf-snmpv3-acm-00.txt, June 1997.

[SNMPv3-USEC] The SNMPv3 Working Group, Blumenthal, U., Wijnen, B. "User-Based Security Model for version 3 of the Simple Network Management Protocol (SNMPv3)", draft-ietf-snmpv3-usec-01.txt, June 1997.

[Localized-Key] U. Blumenthal, N. C. Hien, B. Wijnen "Key Derivation for Network Management Applications"

IEEE Network Magazine, April/May issue, 1997.

[KEYED-MD5] Krawczyk, H.,
"Keyed-MD5 for Message Authentication",
Work in Progress, IBM, June 1995.

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

[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).

APPENDIX A - Installation

A.1. Engine Installation Parameters

During installation, an SNMPv3 engine acting in an authoritative role is configured with several parameters. These include:

(1) 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 of length 44 octets is formed by concatenating digest1, the SNMPv3 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 SNMPv3 engine instantiates the following usecUserEntry in the usecUserTable:

	no privacy support	privacy support
usecUserEngineID	localEngineID	localEngineID
usecUserName	"public"	"public"
usecUserMiId	"public"	"public"
usecUserGroupName	"public"	"public"
usecUserCloseFrom	ZeroDotZero	ZeroDotZero
usecUserAuthProtocol	imfAuthMD5Protocol	imfAuthMD5Protocol
usecUserAuthKeyChange	""	""
usecUserAuthPublic	""	""
usecUserPrivProtocol	none	imfPrivDESProtocol
usecUserPrivKeyChange	""	""
usecUserrivhPublic	""	""
usecUserStorageType	permanent	permanent
usecUserStatus	active	active

A.2. Password to Key Algorithm

The following code fragment demonstrates the password to key algorithm which can be used when mapping a password to an authentication or privacy key. The calls to MD5 are as documented in RFC1321 [RFC1321]

```
void password_to_key(
   u_char *password,    /* IN */
   u_int   passwordlen, /* IN */
   u_char *engineID,    /* IN  - ptr to 12 octet long snmpEngineID  */
   u_char *key)         /* OUT - caller's pointer to 16-byte 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 byte of the password, wrapping  */
          /* to the beginning of the password as necessary.*/
          /*************************************************/
          *cp++ = password[password_index++ % passwordlen];
      }
      MDupdate (&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                  */
   /*****************************************************/
   memcpy(password_buf, key, 16);
   memcpy(password_buf+16, engineID, 12);
   memcpy(password_buf+28, key, 16);

   MD5Init(&MD);
   MDupdate(&MD, password_buf, 44);
   MD5Final(key, &MD);
```

```
   return;
}
```

A.3. Password to Key Sample

The following shows a sample output of the password to key algorithm.

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

Table of Contents