Hands-On SNMPv3 Tutorial & Demo Manual

NuDesign Technologies, Inc.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

Table of Contents

1. INTRODUCTION – WHY SNMPV3 4

1.1 Community based security 4

1.2 User-Based Security (USM) 5
1.2.1 Message Exchanges 6
1.2.1.1 noAuthNoPriv message exchange 6
1.2.1.2 authNoPriv message exchange 7
1.2.1.3 authPriv message exchange 9
1.2.2 Keys 9
1.2.3 User Configuration 10

1.3 View-Based Access Control Model (VACM) 11

2. HANDS-ON SNMPV3 12

2.1 Introduction 12

2.2 Review of Different Modes of Operation 14
2.2.1 noAuthNoPriv 14
2.2.2 authNoPriv 16
2.2.3 authPriv 17
2.2.4 v1/v2c 18
2.2.5 VACM 20

3. DEMO SOFTWARE REFERENCE 24

3.1 TestAgent 24
3.1.1 TestAgent commands 25
3.1.1.1 help or ? 25
3.1.1.2 dumpc[ommunity] 25
3.1.1.3 addc[ommunity] 25
3.1.1.4 delc[ommunity] 26
3.1.1.5 dumpu[sers] 26
3.1.1.6 addu[ser] 26
3.1.1.7 modu[ser] 26
3.1.1.8 delu[ser] 27
3.1.1.9 hex <0 | 1> 27
3.1.1.10 q[uit] 27

3.2 TestManager 27
3.2.1 TestManager commands 28
3.2.1.1 help or ? 28
3.2.1.2 com[munity] 28
3.2.1.3 sec[urityLevel] 28
3.2.1.4 t[arget] 29
3.2.1.5 v[ersion] 29

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

3.2.1.6 con[text] 29
3.2.1.7 us[ername] 29
3.2.1.8 dumpu[sers] 29
3.2.1.9 addu[ser] 29
3.2.1.10 modu[ser] 30
3.2.1.11 delu[ser] 30
3.2.1.12 l[oad] 30
3.2.1.13 un[load] 30
3.2.1.14 g[et] 30
3.2.1.15 sys[descr] 31
3.2.1.16 n[ext] 31
3.2.1.17 s[et] 31
3.2.1.18 w[alk] 31
3.2.1.19 disc[overEngID] 31
3.2.1.20 syn[chronize] 31
3.2.1.21 hex 32
3.2.1.22 q[uit] 32

4. OVERVIEW OF SNMP ARCHITECTURE 33

4.1 SNMP Entities 33
4.1.1 SNMP Manager 33
4.1.2 SNMP Agent 34

4.2 SNMPv3 Applications 35
4.2.1 SNMPv3 Message 36

4.3 Authentication, Privacy Services and Access Control 38
4.3.1 User-Based Security Model 38
4.3.1.1 Overview 38
4.3.1.2 Cryptographic Functions 38
4.3.1.3 Authoritative and Non-Authoritative Engines 39
4.3.1.4 USM Message Parameters 39
4.3.1.5 Timeliness Mechanisms 40
4.3.2 View-Based Access Control 41

5. REFERENCES 43

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

1. Introduction – why SNMPv3

This document describes the usage of TestAgent/TestManager pair of SNMP applications. They are used to demonstrate configuration of SNMPv3 entities and message exchanges between SNMPv3 entities.

SNMPv3 is much more complex than SNMPv1 or SNMPv2c. Complexity comes from the security mechanism used in SNMPv3. SNMPv1 and SNMPv2c provide only a primitive and limited capability for security based on **community** names.

Security model used in SNMPv3 is the User Security Model (**USM**) defined in RFC 2574. USM provides authentication and privacy services for SNMP and it is designed to secure against the following threats:

Modification of Information: An entity could alter an in-transit message generated by an authorized entity in such a way as to effect unauthorized management operations, including the setting of object values.

Masquerade: Management operations that are not authorized for some entity may be attempted by that entity by assuming the identity of an authorized entity.

Message Stream Modification: SNMP is designed to operate over a connectionless transport protocol. There is a threat that SNMP messages could be reordered, delayed, or replayed (duplicated) to effect unauthorized management operations.

Disclosure: An entity could observe exchanges between a manager and an agent and thereby learn the values of managed objects and learn of trap events.

1.1 Community based security

An SNMP community is a relationship between an SNMP agent and a set of SNMP managers that defines authentication, access control and proxy characteristics. The managed system establishes one community for each desired combination of authentication, access control and proxy characteristics. Each community is given a unique (within agent) community name, and the management stations within that community are provided with and must employ the community name in all get and set operations.

Community name is encoded in each SNMPv1/v2c message. A potential intruder can learn the community name easily using packet capture application; v1/v2c messages are not encrypted.

It is obvious that *community based security does not provide protection* against the threats listed above.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

1.2 User-Based Security (USM)

SNMPv3 message consists of three sections (see appendix): msgGlobalData (header), msgSecurityParameters and msgData (scopedPdu). Middle section depends on security model in use. SNMPv3 defines USM as a security model of choice but vendors are free to implements their own security models.

Last field in the message header is an integer that represents security model used for this message. If USM is used this field contains value 3. Header also contains flags field that carry information about security level applied to the message. Possible combinations are:

- noAuthNoPriv – no security applied
- authNoPriv – message is authenticated
- authPriv – message is authenticated and encrypted

Note that encryption without authentication is not valid.

msgSecurityParameters section consists of the six fields that ensure protection against security threats listed above. Note that USM does not prevent Denial of Service and Traffic Analysis attacks.

Not all fields in this section are used in every exchange. Which fields are used depend on the security level.

Two fields that are always used are msgAuthoritativeEngineID (the snmpEngineID of the authoritative SNMP engine involved in the exchange of this message) and msgUserName (the user (principal) on whose behalf the message is being exchanged).

Principal -- is the "who" on whose behalf services are provided or processing takes place. A principal can be, among other things, an individual acting in a particular role; a set of individuals, with each acting in a particular role; an application or a set of applications; and combinations thereof.

SecurityName -- is a human readable string representing a principal. It has a model-independent format, and can be used outside a particular Security Model.

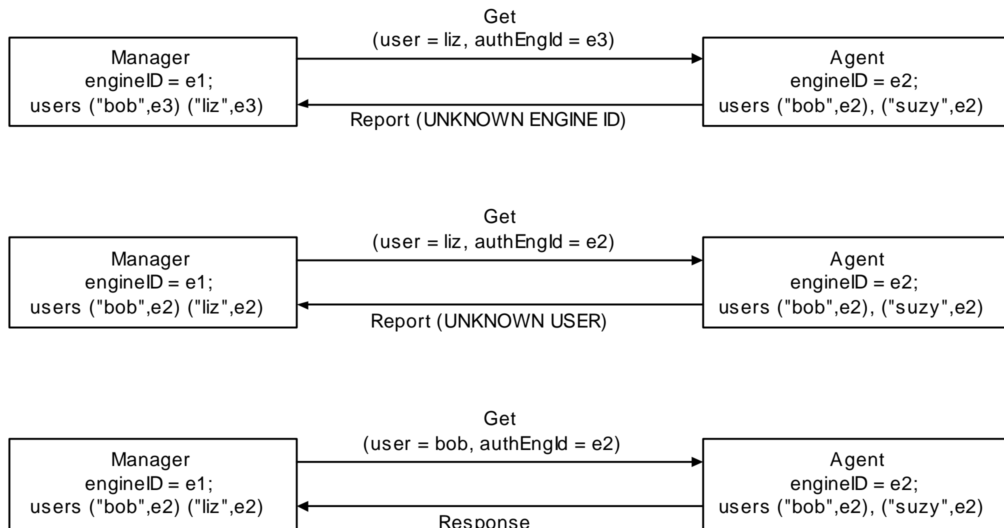
Model-dependent security ID -- is the model-specific representation of a securityName within a particular Security Model. Model-dependent security IDs may or may not be human readable, and have a model-dependent syntax. Examples include **community names**, and **user names**. The transformation of model-dependent security IDs into securityNames and vice versa is the responsibility of the relevant Security Model.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

1.2.1 Message Exchanges

1.2.1.1 noAuthNoPriv message exchange

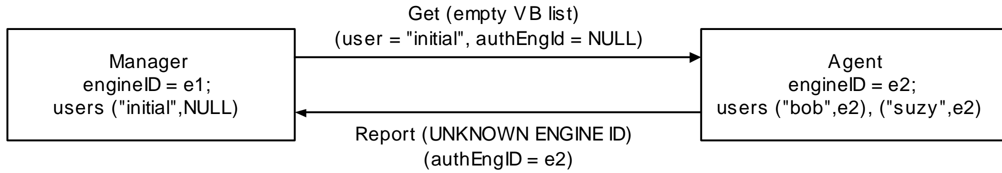
As mentioned above even if no security is applied to the message, two fields from the msgSecurityParameters section must be populated: authoritative engine ID and user name. Manager that wants to retrieve/modify object in the agent has to know that agent's engine id and has to insert into message appropriate user name. Following figure shows possible exchanges between manager and agent in this case (no varbinds are shown, assume that Get asks for the existing MIB object).



In the first case manager sends request with user unknown to the agent. Also authoritative engine id is not that of the agent. Since agent decodes auth engine first and it does not recognize it as its own, agent sends report pdu back to the manager (when agent acts as a "proxy" it checks the list of engineIds on which behalf it operates too). Report pdu is a new pdu in SNMPv3 (actually it was introduced in SNMPv2 but never used, since all the text on usage of report-pdus occurred in security related documents that were subsequently dropped). Report-pdus are used for inter-engine communication.

The first case describes discovery process. In order to communicate with agent, manager has to know that agent's engine ID. RFC states that for discovery manager has to send request with empty varbind list, user "initial" and zero length string as authoritative engine ID. When agent (actually any SNMP entity that can have authoritative role) receives such request, it sends back report-pdu and fills in authoritative engine ID field of the msgSecurityParameters section in the message with its own engine ID. Sender of the message stores received engineID so it can use it in subsequent communication with the agent.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.



RFC 2574 states that authoritative engine ID has to return its own engine ID when it receives "…zero-length, or other illegally sized msgAuthoritativeEngineID…" but then, in the section on processing of outgoing message it states that "…in response or report pdu it MUST always insert its own engine ID…". Most V3 agent vendors follow later statement, which means that ANY request can be used for engine ID discovery.

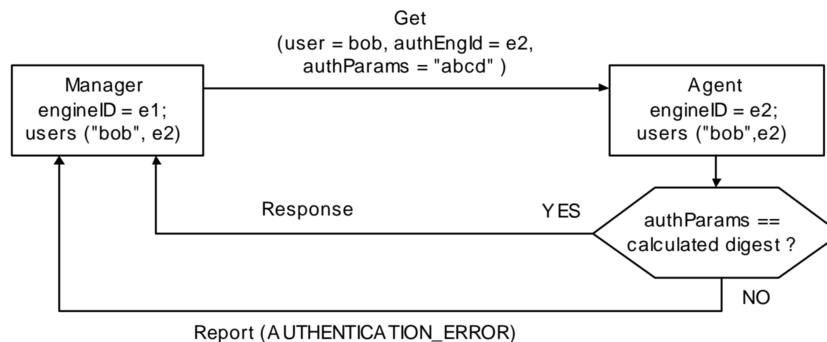
1.2.1.2 authNoPriv message exchange

If we want to assure that the communication between manager and agent is authentic (agent want to be sure that the message came from the manager that it claims it is, and vice versa, manager wants to be sure that the response really came from the target agent), then sender has to calculate message digest (using MD5 or SHA hash functions) and insert it into msgAuthenticationParameters field of the msgSecurityParameters section.

Receiver removes digest from the message, store it into temporary location, fill in removed space with zero octets and calculate message digest. If calculated digest is the same as received digest then message is authentic; otherwise possible impostor is trying to perform illegal operation.

Note that authenticated message does not prevent observer to learn the content of the message.

Following figure shows the authenticated message exchange between manager and agent.



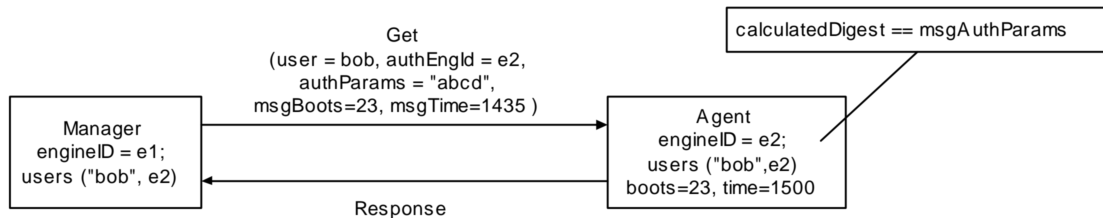
Actually, figure above misses an important part of the authentication. In order to prevent replay attack USM uses timeliness mechanism. Idea is simple. Authoritative engine maintains two objects, **snmpEngineBoots** and **snmpEngineTime**, that refer to local time. Nonauthoritative engine must remain loosely synchronized with each authoritative SNMP engine with which it communicates. For that purpose nonauthoritative engine keeps a local copy of three variables per remote engine ID:

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

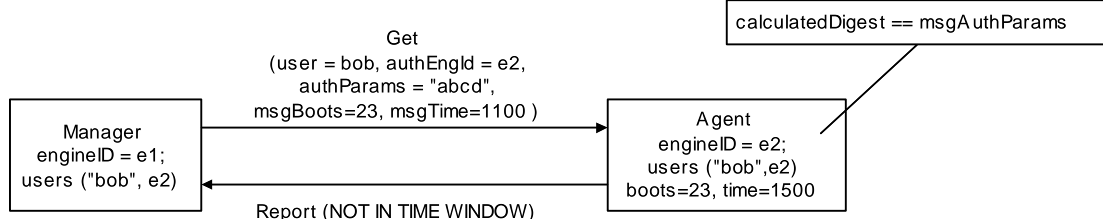
- snmpEngineBoots – of remote engine
- snmpEngineTime – this engine's notion of snmpEngineTime for the remote authoritative engine
- latestReceivedEngineTime – the highest value of msgAuthoritativeEngineTime that has been received by this engine for the remote authoritative engine.

In each authenticated request message (nonauthoritative) entity includes its engine's notion of remote engine's boots and time. When authoritative entity receives authenticated message it checks encoded boots and time values. Boots must match, and time must be within 150 seconds time window. If received message does not satisfy this condition, report-pdu "not in time window" is sent back. Authoritative engine inserts its own boots and time in the report and response message so nonauthoritative engine can update local copies of these values. This is illustrated in the following figure.

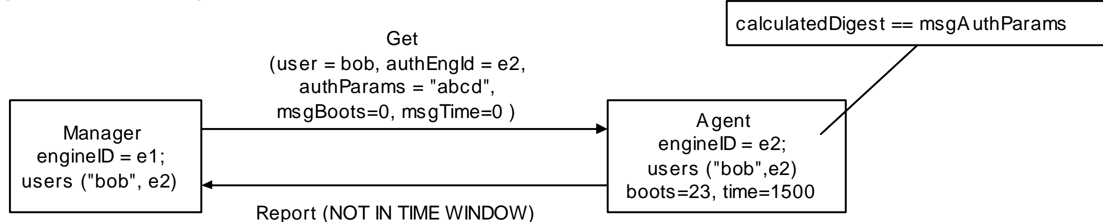
authenticated message within time window
(msgBoots==boots and |time-msgTime|<150)



authenticated message not in time window
(msgBoots==boots and |time-msgTime|>150)



Synchronization
(msgBoots==0 and msgTime==0)

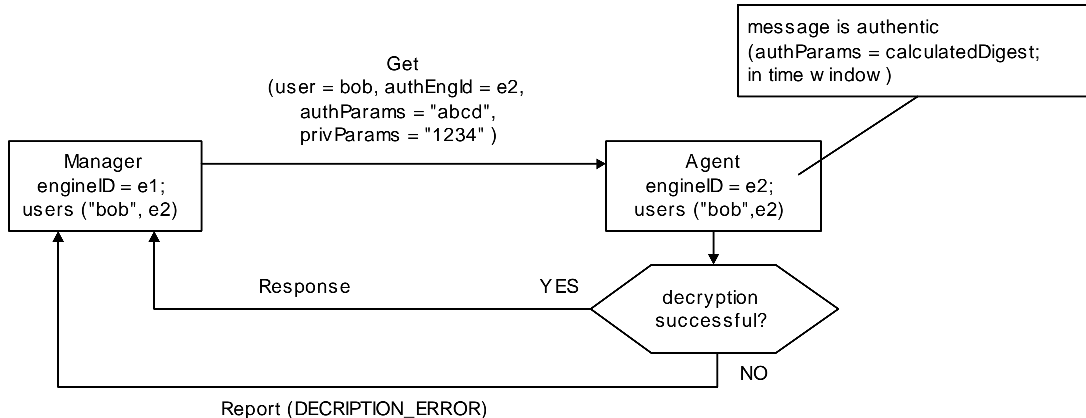


This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

1.2.1.3 authPriv message exchange

If we want to assure that the communication between manager and agent is protected from disclosure, an encryption must be applied. Not whole message is encrypted, but only scoped pdu. The algorithm of choice for encryption in SNMPv3 is the cipher block chaining (CBC) mode of the Data Encryption Standard (DES). Vendors are free to use any other algorithm.

Note that if encryption is applied, message must be authenticated as well.



1.2.2 Keys

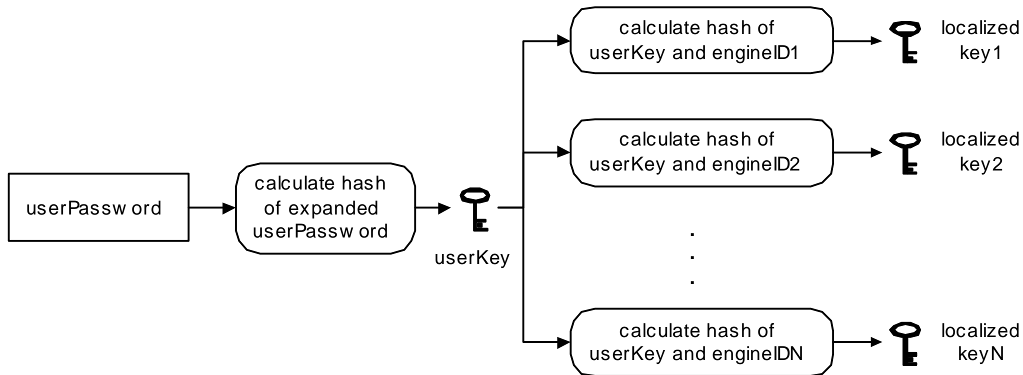
For both authentication and encryption SNMP entity requires appropriate key. Since all cryptographic algorithms used are symmetric, both sides must have the same key.

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

In order to simplify key deployment SNMPv3 proposes PasswordToKey algorithm. SNMP entity calculates key from the password using specified hash function. For the same password this function will create the same key. To make things simple, yet as secure as possible, calculated key is “localized”. I.e. authoritative engine id is “wrapped” with calculated key and hash function is applied on that octet string. This ensures that the same password produce different key for the different engine Ids. If key is stolen for one entity this will still not break security of the other entities.

Key localization enables a principal to share a unique authentication and encryption key with each remote engine while only maintaining a single authentication and encryption key locally.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.



1.2.3 User Configuration

As we saw, in order to exchange messages between manager and agent both parties have to be configured with at least one and the same user. Also manager has to know/discover remote engine id. If authenticated communication is desired, both authentication algorithm (MD5, SHA) and authenticationKey, must be known to both parties. For encryption two more pieces of information are necessary: crypto algorithm (DES) and encryptionKey.

User configuration resides in Usm User table.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

1.3 View-Based Access Control Model (VACM)

Now, if only security in SNMPv3 is what is described above, there will be a problem. If we have users that we want to assign different level of privileges (for example, we want to allow administrator to reset/reboot remote device while we want to prevent ordinary user from doing that (at the same time we want to allow every user to read status of the remote agent) it is not enough just to categorize users by giving them authentication and privacy passwords/keys.

This is where VACM comes into the picture.

VACM uses a MIB to specify which user can access which part of the agent MIB under specified conditions. Conditions include **security level** (for example certain part of the MIB can be accessed only using authenticated requests, the other only if request is both authenticated and encrypted, …), **security model** (for example we do not want to allow SNMPv1 or SNMPv2 managers to access v3 configuration tables), **userName** (for example “bob” can access all objects in the MIB, while “liz” can access only mib-2 subtree), **viewType** (for example user can be allowed to “read” but not “write” to certain object), and which **context** the object exist.

VACM does not specify access rights for every single instance in the agent’s MIB. Rather it specifies them per subtree. **Subtree** is a set of all objects and object instances that have a common object identifier prefix to their names.

So far we assumed that user (that the agent knows of) could access all objects in the MIB. In this scenario multilingual agent is not protected at all, since we can use SNMPv1 or v2 manager to retrieve/modify all objects. If you look in the NVAgentCfg1.txt file you can see that VacmViewTreeFamilyTable contains single row with viewName = “everything” and subtree = “1.3.6.1”. In other words, for examples above, we allowed user “bob” to access all MIB instances in the agent.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

2. Hands-On SNMPv3

2.1 Introduction

We'll start exercise with (almost) empty configuration files. The only info stored initially is EngineID. If this parameter is not present, agent and manager will create engineId the first time they are started and this info will be stored in the NV ram (i.e. in file). For convenience both application will be running on the same machine.

First start TestAgent (note that the command line argument is the name of the NV storage file for the agent; if it is not present agent will try to use one with name "NVAgCfg.txt" if it exists, if it does not exist, an empty configuration file will be created).

```
C:\temp> TestAgent
****************************************************
* NuDesign Team Multilingual (v1/v2c/v3) SNMP Agent
*
* Version:          SNMPv3
* Bound to:
*       Address[1] 127.0.0.1:161 [teListener]
*       Address[2] 192.168.1.51:161 [teListener]
* Community table has 1 row(s)
        [1] = (public, readCreate)
****************************************************

* v3 engineId=00:02:34:56:12:34:56:78:00:bb:bb:00
* v3 boots=1

* USM USER TABLE has 0 row(s)
[NDSnmpAg]#
```

As we can see, this agent claims multilingual SNMP support, prints configuration parameters (community, engineId, etc.) and waits with the prompt `[NDSnmpAg]#` for commands. Type "?" or "help" for the list of available commands. It also waits for the SNMP requests by listening on port 161. Note that the transport is bound to each particular address. In case machine has multiple network cards this will allow us to determine the actual interface that the request came through.

The USM user table is empty. TestAgent implements NDTestV2-MIB, and also mib-2 system and snmp group, and SNMPv3 MIBs (defined in rfc2571 through rfc2575).

Now start TestManager (note that the command line argument is the name of the NV storage file for the manager; if it is not present manager will try to use one with name "NVMgrCfg.txt" if it exists, if it does not exist, an empty configuration file will be created)

```
C:\temp> TestManager
Loading rfc1213.mib
Loading rfc1907.mib
Loading rfc2571.mib
```

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

```
Loading rfc2572.mib
Loading rfc2573.mib
Loading rfc2574.mib
Loading rfc2575.mib
Loading NDTestV2.mib
*******************************************************
* NuDesign Team Multilingual (v1/v2c/v3) SNMP Manager
*
* Version:          SNMPv3
*
* Bound to:
*       Address[3] 127.0.0.1:162 [teListener]
*       Address[4] 192.168.1.51:162 [teListener]
*******************************************************
* v3 engineId=00:00:12:99:7f:00:00:01:87:65:43:21
* v3 boots=0
 [NDSnmpMgr]#
```

As you can see it starts almost identically as the agent (the prompt is different). TestManager is multilingual SNMP manager (v1/v2c/v3). On starting it tries to load rfc1213.mib, rfc1907.mib, NDTestV2.mib, and v3 mibs (rfc2571 through rfc2575). While manager works without any MIB loaded, it is more convenient to have them loaded since you can then use object names instead of “raw” oids for requests. MIBs must be in the same directory where TestManager resides.

On TestManager’s prompt `[NDSnmpMgr]#`“ type “?” or “help” for the list of available commands.

Type “dumpusers” command on the manager prompt. This command will print UsmUser table. Since we don’t have any user configured, we should get:

```
USM USER TABLE has 0 row(s)
```

message. Type “t” command. This will print current target, i.e. manager will send all subsequent requests to the destination set as a current target. When we want to send request to the other agent, we have to change target first. We should get:

```
Address: 127.0.0.1:161
Community: public
Timeout: 5
NumRetries: 0
```

Since agent is running on the same machine as manager we’ll leave target as it is.
Verify that the current version is v3:

```
[NDSnmpMgr]# v
Current version is SNMPv3
```

and that the current security level is noAuthNoPriv:

```
[NDSnmpMgr]# sec
Security level is 1[noAuthNoPriv]
```

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

2.2 Review of Different Modes of Operation

2.2.1 noAuthNoPriv

The first thing to do is to retrieve value of the remote engineID (agent's engineID). We invoke "disc" command (for each command you can type full name or up to that many letters that will make command unique; for example "disc", "disco", "discovery" are all the same command).

Agent will respond with Report UNKNOWN_ENGINE_ID. Manager's engine stores address / engineID pair. Also, manager stores received engineID; it will be used for adding users to the UsmUser table shortly.

Now try:

```
[NDSnmpMgr]# get sysDescr.0
```

An error message is printed: UNKNOWN USER NAME. This is the error from the manager's SNMP engine. In order to send request, engine consults UsmUser table to get necessary message parameters. Since UsmUser table is empty, an error is returned. Let's add user:

```
[NDSnmpMgr]# addusers bob
```

This command will add user "bob" to the UsmUser table. Since only user name is specified, engineID for this user is the last discovered engineID (see above). Syntax for the "adduser" command is

```
addu[ser] <name> [<authProt> <authPswd> [<privProt privPswd>]]
```

authentication and privacy protocol are set to "none" and passwords are not used.

Now, we have a valid user (you can see it using "dumpusers" command). Change the current user to "bob" using.

```
[NDSnmpMgr]# user bob
```

Try again:

```
[NDSnmpMgr]# get sysDescr.0
```

This time, we should get response from the agent: report UNKNOWN_USER_NAME. It is not enough to add user just to the manager's table. Agent has to know about the user as well. Now switch to the agent's window and type:

```
[NDSnmpAg]# addusers bob
```

This command adds user "bob" to the agent's UsmUser table. EngineID used for "bob" is the local engine ID. UsmUser table has two index columns: user name and engine id. For manager engineID is the one of the remote (agent's) engine ID, for agent it is the local one.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

Invoke “dumpusers” commands on agent and on manager prompt. Both should result with the following lines:

```
* USM USER TABLE has 1 row(s)
1: bob; 00:00:12:99:7F:00:00:01:aa:aa:aa:aa; authProt=none; privProt =
none
```

Now, that the both sides know about the same user, we can successfully retrieve sysDescr, right? Try:

```
[NDSnmpMgr]# get sysDescr.0
```

Agent responds with authentication error. What is wrong? The answer is: VACM is not configured. When agent receives request from the valid user, before sending response it checks whether requested instance lies in the view. If it is outside the view for that user an authentication error is returned. In case request is GetNext, all instances not in the view are skipped. We have to manually configure agent’s NV storage for the VACM. Close the agent (type “q” on the prompt) and open NVAgCfg.txt in the text editor. Add bold text to the following sections in the file:

```
[VacmSecurityToGroupTable]
; secModel secName groupName storageType rowStatus
1=USM(3) bob grpAll nonVolatile(3) active(1)

[VacmAccessTable]
;grpName cntxtPrefix secModel secLevel contextMatch readViewName
writeViewName notifyViewName storageType rowStatus
1=grpAll "" USM(3) noAuthNoPriv(1) prefix(2) all all all nonVolatile(3)
active(1)

[VacmViewTreeFamilyTable]
; viewName subtree familyMask familyType storageType rowStatus
1=all 1.3.6.1 FF included(1) nonVolatile(3) active(1)
```

One more table is part of the VACM: VacmContextTable. For this tutorial we will be working only with default context (name of such context is zero length string) so we’ll leave context table empty.

Let’s explain the three tables above. In VacmSecurityToGroupTable we specified that user “bob” will be using “USM” security model and he belongs to group “grpAll”. The VacmAccess Table specifies that group “grpAll” requires “USM” security model, that the context is default (zero length string), and that security level for this group is minimal (no AuthNoPriv). Security level tells us that this group can be accessed with any valid security level. Also, this table specifies view names for this group (for read, write or notify operations).

Finally, VacmViewTreeFamilyTable maps view names to subtrees and also specify whether this subtree is accessible for that view (“included”) or not (“excluded”). In our example, view name “all” has access (“included”) to the subtree “1.3.6.1”, in other words to all objects and instances in the agent’s MIB.

Save the file and restart the agent. Note that we do not have to add user “bob” again. Agent saved that information from previous run.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

Now try:

```
[NDSnmpMgr]# get sysDescr.0
```

This time agent should respond with:

```
SysDescr.0[DispStr]: NuDesign SNMP Agent
```

2.2.2 authNoPriv

Let's try authenticated message exchange. Change security level in manager to authNoPriv:

```
[NDSnmpMgr]# sec 2
Security level is set to 2[authNoPriv]
```

Now try:

```
[NDSnmpMgr]# get sysDescr.0
```

This time an error is returned from local engine for reasons of UNSUPPORTED SECURITY LEVEL. Manager's engine consulted UsmUser table and found that user "bob" is not configured to use authentication. Lets change bob's parameters:

```
[NDSnmpMgr]# moduser bob MD5 md5authpswd
```

Invoke

```
[NDSnmpMgr]# dumpusers
* USM USER TABLE has 1 row(s)
1: bob; 00:00:12:99:7F:00:00:01:aa:aa:aa:aa; authProt=MD5; privProt =
none
```

We can see that "bob" is configured to use authentication with MD5 protocol. Now try:

```
[NDSnmpMgr]# get sysDescr.0
```

response comes from the agent is: UNSUPPORTED SECURITY LEVEL. Agent still has user "bob" configured as noAuthNoPriv user. Repeat the same modification command on the agent side:

```
[NDSnmpAg]# moduser bob MD5 md5authpswd
```

Now response to

```
[NDSnmpMgr]# get sysDescr.0
```

will be report NOT IN TIME WINDOW. For authenticated message exchange, to prevent replay attack, SNMPv3 uses timeliness mechanism. Report contains valid agent's time and boots values and local manager's engine is supposed to store them for use in the next transactions.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

Repeat:

```
[NDSnmpMgr]# get sysDescr.0
```

This time everything should be fine; response is

```
SysDescr.0[DispStr]: NuDesign SNMP Agent
```

Note that we have used passwords for authentication. Engine internally calculates key from the password. What happens if the password is wrong? Let's try:

```
[NDSnmpAg]# moduser bob MD5 md5wrongauthpswd
```

Now try:

```
[NDSnmpMgr]# get sysDescr.0
```

and agent responds with AUTHENTICATION_ERROR.

2.2.3 authPriv

Lets check the message exchange with full privacy. Modify user on both sides:

```
[NDSnmpMgr]# moduser bob MD5 md5authpswd DES desprivpswd
```

and

```
[NDSnmpAg]# moduser bob MD5 md5authpswd DES desprivpswd
```

Change security level on the manager side:

```
[NDSnmpMgr]# sec 3
Security level is set to 3[authPriv]
```

Now try:

```
[NDSnmpMgr]# get sysDescr.0
```

and agent responds with

```
SysDescr.0[DispStr]: NuDesign SNMP Agent
```

Modify user on the manager side:

```
[NDSnmpMgr]# moduser bob MD5 md5authpswd DES deswrongprivpswd
```

Note that now the privacy password is different.

Now try:

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

```
[NDSnmpMgr]# get sysDescr.0
```

and timeout is reported. What happened? Manager has sent encrypted request using one key, while agent decrypted the same message using different key. There will be no error in decryption (except decrypted scopedpdu won't resemble original). Agent tries to decode such scoped pdu and decoding fails. ParsingErrors counter is incremented and agent does not respond. Note that report DECRYPTION_ERROR will not be sent at this time since received message has valid encrypted scoped pdu (unfortunately with wrong key but agent does not know that).

2.2.4 v1/v2c

Lets examine v1 and v2 requests (both agent and manager are multilingual). We'll try v1 first. Change current version in manager to v1:

```
[NDSnmpMgr]# v 1
Version is set to SNMPv1
```

Verify that community string is "public":

```
[NDSnmpMgr]# target
Address: 127.0.0.1:161
Community: public
Timeout: 5
NumRetries: 0
```

Now try:

```
[NDSnmpMgr]# get sysDescr.0
```

and agent sends error response back: varbind at index 1 caused authentication error. What is wrong? Again, access control module is responsible. VACM currently knows only about security name "bob" (the default transformation of the User-based Security Model dependent security ID (username) to the securityName and vice versa is the identity function so that the securityName is the same as the userName.) which is supposed to use USM security model.

SNMPv1 security model is community based and securityName in the message is community string "public". There is no entry in the VacmSecurityToGroupTable for securityName "public" and processing of this request failed with authentication error.

Close the agent (type "q" on the prompt) and open NVAgCfg.txt in the text editor. Add bold text to the VacmSecurityToGroupTable section in the file:

```
[VacmSecurityToGroupTable]
; secModel secName groupName storageType rowStatus
1=USM(3) bob grpAll nonVolatile(3) active(1)
2=SNMPv1(1) public grpAll nonVolatile(3) active(1)
3=SNMPv2c(2) public grpAll nonVolatile(3) active(1)
```

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

What applies to SNMPv1 regarding security, applies to SNMPv2c as well, since both use community based security model. For that reason we added the line for v2 at this time as well.

Note that security model for the securityName “public” is in one case SNMPv1 and in the other SNMPv2c. Having that information, SNMP engine will process SNMPv1 and SNMPv2c messages with community based security subsystem module.

This is not enough. We have to update VacmAccessTable as well:

```
[VacmAccessTable]
;grpName cntxtPrefix secModel secLevel contextMatch readViewName
writeViewName notifyViewName storageType rowStatus
1=grpAll "" USM(3) noAuthNoPriv(1) prefix(2) all all all nonVolatile(3)
active(1)
2=grpAll "" SNMPv1(1) noAuthNoPriv(1) prefix(2) all all all
nonVolatile(3) active(1)
3=grpAll "" SNMPv2c(2) noAuthNoPriv(1) prefix(2) all all all
nonVolatile(3) active(1)
```

The 2nd and 3rd entry tell the agent that group “grpAll” can be accessed with requests that use SNMPv1 and SNMPv2c security model. In other words we’ve configured our agent to respond to v1, v2 and v3 requests, that “grpAll” has access to the all objects in the agent’s MIB (note that v3 user “bob” maps to “grpAll”, and that v1/v2 community “public” maps to that group as well) and that minimal security level necessary to access objects in the MIB is noAuthNoPriv.

Restart agent and try (version is still v1)

```
[NDSnmpMgr]# get sysDescr.0
```

This time agent should respond with:

```
SysDescr.0[DispStr]: NuDesign SNMP Agent
```

Try the same in v2:

```
[NDSnmpMgr]# v 2
Version is set to SNMPv2c
[NDSnmpMgr]# get sysDescr.0
```

agent should respond with:

```
SysDescr.0[DispStr]: NuDesign SNMP Agent
```

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

2.2.5 VACM

Someone will ask after this little exercise: Why use v3 since agent responds to v1 and v2? Why go through all trouble of v3 security (authentication and encryption) if every object in the agent can be accessed using v1 or v2? Well, this is the point where full power of VACM comes into the focal point.

In the example above we've basically created only one view to our MIB.

VacmViewTreeFamilyTable has only one entry that tells the agent that all objects under the "internet" subtree are included into the "all" view. From the access table we know that the group "grpAll" is configured for view "all" for all security models (SNMPv1, SNMPv2c and USM).

VacmSecurityToGroupTable tells us that both v3 user "bob" (again username and securityName are used interchangeably) and v1/v2 community "public" (community name is interpreted as securityName) belong to group "grpAll".

Lets make our agent secure. Assume that we do not want to allow v1/v2 users to access our enterprise specific MIB (agent implements NDTestV2-MIB). VACM configuration should look like (bold text is what should be changed, do not forget to shutdown agent before editing config file, otherwise agent will overwrite it next time it stops):

```
[VacmSecurityToGroupTable]
; secModel secName groupName storageType rowStatus
1=USM(3) bob grpAll nonVolatile(3) active(1)
2=SNMPv1(1) public grpRestricted nonVolatile(3) active(1)
3=SNMPv2c(2) public grpRestricted nonVolatile(3) active(1)


[VacmAccessTable]
;grpName cntxtPrefix secModel secLevel contextMatch readViewName
writeViewName notifyViewName storageType rowStatus
1=grpAll "" USM(3) noAuthNoPriv(1) prefix(2) all all all nonVolatile(3)
active(1)
2= grpRestricted "" SNMPv1(1) noAuthNoPriv(1) prefix(2) restricted
restricted restricted nonVolatile(3) active(1)
3= grpRestricted "" SNMPv2c(2) noAuthNoPriv(1) prefix(2) restricted
restricted restricted nonVolatile(3) active(1)


[VacmViewTreeFamilyTable]
; viewName subtree familyMask familyType storageType rowStatus
1=all 1.3.6.1 FF included(1) nonVolatile(3) active(1)
2=restricted 1.3.6.1 FF included(1) nonVolatile(3) active(1)
3=restricted 1.3.6.1.4.1.4761 FF excluded(2) nonVolatile(3) active(1)
```

We've changed 2nd and 3rd row in both VacmSecurityToGroupTable and VacmAccessTable. Now "public" community for both v1 and v2 security model belongs to group "grpRestricted". From VacmAccessTable we removed v1/v2 from "grpAll" and added "grpRestricted" for v1/v2 with new view name "restricted".

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

The most interesting part is VacmViewTreeFamilyTable. The 2nd row is identical to the 1st except for the viewName (“restricted” as opposed to “all” in the 1st row). If there was no 3rd row this configuration would be exactly the same as before: v1/v2 could access everything. Since we wanted to restrict v1/v2 users from accessing our private MIB implementation (subtree 1.3.6.1.4.1.4761) we’ve added 3rd row that specify that subtree 1.3.6.1.4.1.4761 is **excluded.**

The VACM module in the SNMP engine always uses the subtree with the most matching subids when making decision should it allow or not access to target object,

Restart agent. Change the version in manager to v1 and try:

```
[NDSnmpMgr]# get sysDescr.0
```

agent should respond with:

```
SysDescr.0[DispStr]: NuDesign SNMP Agent
```

which is OK, system group is still included into the view. Now try:

```
[NDSnmpMgr]# get tvModel.0
```

which is the object from the NDTestV2.mib. Agent responds with authentication error. Repeat the same in v2. The behaviour is the same. Change the version to v3 (user is “bob”) and now reponse to

```
[NDSnmpMgr]# get tvModel.0
```

is

```
tvModel.0[DispStr]: CTV 21
```

What about GetNext? For GetNext excluded subtrees are the same as if they do not exist. Returned value for the GetNext that resolves in instance in excluded region will be the first instance in the first included region after the first matched “next” instance or EndOfMIB if such instance does not exist.

Configuring VACM is not restricted to v1/v2 only. We will now add USM user “liz” to our agent and manager.

```
[NDSnmpMgr]# adduser liz
```

and

```
[NDSnmpAg]# adduser liz
```

These commands will create user “liz” with no authorization or privacy protocol configured. We want to allow “liz” to read all the objects from the agent, but we do not want to allow her to modify any. We also want to force user “bob” to use encryption in communication with the agent. This is the new look of the VAM configuration in the NVAgCfg.txt:

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

```
[VacmSecurityToGroupTable]
; secModel secName groupName storageType rowStatus
1=USM(3) bob grpAll nonVolatile(3) active(1)
2=SNMPv1(1) public grpRestricted nonVolatile(3) active(1)
3=SNMPv2c(2) public grpRestricted nonVolatile(3) active(1)
4=USM(3) liz grpReadOnly nonVolatile(3) active(1)

[VacmAccessTable]
;grpName cntxtPrefix secModel secLevel contextMatch readViewName
writeViewName notifyViewName storageType rowStatus
1=grpAll "" USM(3) authPriv(3) prefix(2) all all all nonVolatile(3)
active(1)
2=grpRestricted "" SNMPv1(1) noAuthNoPriv(1) prefix(2) restricted
restricted restricted nonVolatile(3) active(1)
3=grpRestricted "" SNMPv2c(2) noAuthNoPriv(1) prefix(2) restricted
restricted restricted nonVolatile(3) active(1)
4=grpReadOnly "" USM(3) noAuthNoPriv(1) prefix(2) all "" ""
nonVolatile(3) active(1)

[VacmViewTreeFamilyTable]
; viewName subtree familyMask familyType storageType rowStatus
1=all 1.3.6.1 FF included(1) nonVolatile(3) active(1)
2=restricted 1.3.6.1 FF included(1) nonVolatile(3) active(1)
3=restricted 1.3.6.1.4.1.4761 FF excluded(2) nonVolatile(3) active(1)
```

Again, lines in bold are changed. Note that VacmSecurityToGroupTable has new row that specify USM user "liz" that belong to group "grpReadOnly". VacmAccessTable has new row too; it specifies that the group "grpReadOnly" requires noAuthNoPriv security and only one view name is specified: readViewName. The writeViewName and notifyViewName are zero length strings which means no view for write nor notify operations are configured for "grpReadOnly". In other words, SNMP set is disabled for members of this group ("liz"). Note also that 1st row has been changed: "grpAll" now requires minimal security level to be authPriv (i.e both authorization and encryption must be applied to the message).

Restart the agent and try:

```
[NDSnmpMgr]# user bob
[NDSnmpMgr]# securityLevel 1
Security level is set to 1[noAuthNoPriv]
[NDSnmpMgr]# get sysDescr.0
```

the agent responds with authentication error. Reason: we've configured VACM such that user bob MUST use auhPriv level of security. Change security level to auhPriv.

```
[NDSnmpMgr]# securityLevel 3
Security level is set to 3[authPriv]
```

Try again

```
[NDSnmpMgr]# get sysDescr.0
```

and agent now responds with

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

```
SysDescr.0[DispStr]: NuDesign SNMP Agent
```

Now change current user to “liz” and security level to noAuthNoPriv.

```
[NDSnmpMgr]# user liz
[NDSnmpMgr]# securityLevel 1
Security level is set to 1[noAuthNoPriv]
```

Now try:

```
[NDSnmpMgr]# get sysDescr.0
```

agent responds with

```
SysDescr.0[DispStr]: NuDesign SNMP Agent
```

Lets see what happens when “liz” tries to modify object in the agent:

```
[NDSnmpMgr]# set sysContact.0 liz
```

agent responds with authentication error. If you look in the agent window you can see message from the VACM: NO SUCH VIEW. As we wanted grpReadOnly has no “write” view configured and accordingly set failed. Note that if you change version to v1 set sysContact will succeed since sysContact is in the “write” view for restricted group.

Combinations are endless, so please feel free to experiment as much as possible. Happy SNMPv3-ing.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

3. Demo Software Reference

3.1 TestAgent

TestAgent is multilingual SNMP agent (v1/v2c/v3). It implements NDTestV2-MIB. Here is the tree representation of this MIB:

```
. . . [ 1] enterprises
        |
        +- [4761] nuDesignTeam
           |
           +- [ 1] software
           +- [99] ndtExperimental
              |
              +- [ 1] tv
              |  |
              |  +- [ 1] -RO- DisplayString    tvManufacturer
              |  +- [ 2] -RW- DisplayString    tvModel
              |  +- [ 3] -RW- OctetString      tvModelId
              |  +- [ 4] -RW- Enum             tvControl
              |  +- [ 5] -RW- Enum             tvChannel
              |  +- [ 6] -RW- Integer32        tvVolume
              |  +- [ 7] -RO- ObjectIdentifier tvLockCode
              |  +- [ 8] -RO- TimeTicks        tvUpTime
              |  +- [ 9] -RW- Counter32        tvNumTimesTurnedOn
              |  +- [10] -RW- IpAddress        tvIpAddr
              |  +- [11] -RW- Bits             tvColorBits
              |  +- [12] -RO- Counter64        tvInFrames
              |
              +- [ 2] houseAlarm
                 |
                 +- [ 1] -RO- DisplayString    houseLocation
                 +- [ 2] -RO- INTEGER          houseNumAlarmSensors
                 +- [ 3] alarmSensorTable
                 |  |
                 |  +- [ 1] alarmSensorEntry
                 |     |
                 |     +- [ 1] -~ Integer32        alarmSensorFloor
                 |     +- [ 2] -~ Integer32        alarmSensorIndex
                 |     +- [ 3] -RW- DisplayString  alarmSensorDescr
                 |     +- [ 4] -RO- Enum           alarmSensorType
                 |     +- [ 5] -RW- Enum           alarmSensorState
                 |     +- [ 6] -RO- Counter32      alarmSensorCount
                 |     +- [ 7] -RO- OctetString    alarmSensorId
                 |     +- [ 9] -RC- RowStatus      alarmSensorRowStatus
                 |
                 +- [ 4] -RO- DisplayString    housePhoneNumber
```

Also, internally SNMP engine implements mib-2 system and snmp groups, and v3 MIBs (rfc2571 through rfc2575).

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

3.1.1 TestAgent commands

3.1.1.1 help or ?

This command will print all available commands. Printout looks like:

```
 Action                  Command
--------------     ------------------------------------------------------------
--
 Help                    help | ?
 Dump Community          dumpc[ommunity]
 Add Community           addc[ommunity] name maxaccess
                         maxaccess = 1-notify;2-readOnly;3-readwrite;4-readcreate
 Delete Community        delc[ommunity] name
 Dump Usm Table          dumpu[sers]
 Add USM User            addu[ser] <name>
                         [<authProt><authPswd>[<privProt><privPswd>]]
                         authProt = MD5|SHA  privProt = DES
 ModifyUSM User          modu[ser] <name>
                         [<authProt><authPswd>[<privProt><privPswd>]]
                         authProt = MD5|SHA  privProt = DES
 Delete USM User         delu[ser] <name>
 DumpHex                 hex <0 | 1>
 Quit                    q
```

3.1.1.2 dumpc[ommunity]

This command prints current content of the agent's community table. Format is:

"[" # "] = (" <name> "," <maxAccess> ")"

where # is number, name is community string and maxAccess is max access allowed for this community. For example:

```
* Community table has 1 row(s)
        [1] = (public, readCreate)
```

Which means that "public" community has maximal rights.

3.1.1.3 addc[ommunity]

This command adds new community (or modifies existing with the same name). Syntax is:

```
addc[ommunity] name maxaccess
```

For example

```
[NDSnmpAg]# addc private 2
```

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

will add “private” community name with max access equal to readOnly.

3.1.1.4 delc[ommunity]

This command removes community with the name specified as argument. Syntax is:

```
delc[ommunity] name
```

For example

```
[NDSnmpAg]# delc private
```

will remove “private” community from the agent’s list.

3.1.1.5 dumpu[sers]

This command prints current content of the USM User table. Format is:

```
# ": " <name> ";"<engineID> ";" <authProtocol> ";" <privProtocol> ";"
```

where <name> is username, <engineID> is localEngineID, <authProtocol> could be “MD5” or “SHA” and <privProtocol> could only be “DES”.

3.1.1.6 addu[ser]

This command adds new USM user. Syntax is:

```
addu[ser] <name> [<authProt><authPswd>[<privProt><privPswd>]]
```

The only mandatory argument is <name>. If other arguments are not supplied user with no authProtocol and no privProtocol will be created. If authProt is present, authPswd must be present as well. The same is true for privacy parameters.

3.1.1.7 modu[ser]

This command modifies existing USM user. Syntax is:

```
modu[ser] <name> [<authProt><authPswd>[<privProt><privPswd>]]
```

Arguments are the same as for adduser.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

3.1.1.8 delu[ser]

This command removes existing USM user. Syntax is:

```
delu[ser] <name>
```

3.1.1.9 hex <0 | 1>

This command enables (hex 1) or disables (hex 0) printing of raw packets (and some extra values like keys, initial vectors, …). By default hex is off.

3.1.1.10 q[uit]

This command shuts down the agent and exits the application.

3.2 TestManager

TestManager is multilingual SNMP manager (v1/v2c/v3). On starting it tries to load rfc1213.mib, rfc1907.mib, NDTestV2.mib, and v3 mibs (rfc2571 through rfc2575). While manager works without any MIB loaded, it is more convenient to have them loaded since then you can use object names instead or “raw” oids. MIBs must be in the same directory where TestManager resides.

TestManager can send Get, GetNext and Set requests. The Walk command is available too. It performs GetNext operations in succession starting from the “org” to the end of MIB or till specified number of objects are retrieved.

Request is sent to the agent currently configured as target (see “target” command). Version of the request message is set to the current version (see “version” command).

For v1 and v2c messages, current community string is used (it can be changed using “community” or “target” commands).

A few parameters are necessary for the v3 messages. Again, manager uses currently set securityLevel (see “securityLevel” command), user (see “user” command) and context name (see “context” command). Initially context name is set to default (zero length string). Note that if you change context name, make sure that the target agent recognize it. There is no provision for changing contextEngineId since TestAgent does not have proxy functionality, contextEngineId is always set to that of the target engineId.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

3.2.1 TestManager commands

3.2.1.1 help or ?

This command will print all available commands. Printout looks like:

```
  Action              Command
--------------  ------------------------------------------------------------
--
 Help                help | ?
 Target              t[arget] [<udpAddr> [<community> <timeout> <numRetries>]]
 Community Name      com[munity] [<name>]
 Get                 g[et] <oid> ...
 GetNext             n[ext] <oid> ...
 Set                 set <oidName, value>
 Walk                w[alk] <count>
 Stop Walk           w
 GetSysDesc          sys[desc]
 Version             v[ersion] [<1|2|3>]
 DiscoverEngId       disc[overEngID]
 Sync With Eng       syn[chronize] [<securityName>]
 Security Level      sec[urityLevel] <1|2|3>
 User Name           us[ername] <name>
 Add USM User        addu[ser] <name> [<authProt> <authPswd> [<privProt
                     privPswd>]]
 ModifyUSM User      modu[ser] <name> [<authProt> <authPswd> [<privProt
                     privPswd>]]
 DeleteUSMUser       delu[ser] [<name>]
 Dump Usm Table      dumpu[sers]
 Context Name        con[text] [<name>]
 Load MIB            l[oad] <mib-file-name>
 Unload MIB          un[load] <mib-module-name>
 DumpHex             hex <0 | 1>
 Quit                q
```

3.2.1.2 com[munity]

Without parameters this command prints current community string. If parameter is present community will be changed. Syntax is:

```
com[munity] [<name>]
```

3.2.1.3 sec[urityLevel]

Without parameters this command prints current security level. If parameter is present security level will be changed. Syntax is:

```
sec[urityLevel] <1|2|3>
```

where 1 = noAuthNoPriv, 2=authNoPriv and 3=authPriv.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

3.2.1.4 t[arget]

Without parameters this command prints current target. If parameters are present target will be changed. Syntax is:

```
t[arget] [<udpAddr> [<community> <timeout> <numRetries>]]
```

where <udpAddr> is UDP address (for example: 192.168.1.1:161), <community> is community string, <timeout> is timeout in seconds and <numRetries> is number of retries for requests.

3.2.1.5 v[ersion]

Without parameters this command prints current SNMP version. If parameter is present SNMP version will be changed. Syntax is:

```
v[ersion] [<1|2|3>]
```

where 1 = SNMPv1, 2=SNMPv2c and 3=SNMPv3.

3.2.1.6 con[text]

Without parameters this command prints current context name. If parameter is present context name will be changed. Syntax is:

```
con[text] <name>
```

3.2.1.7 us[ername]

Without parameters this command prints current USM user. If parameter is present user will be changed. Syntax is:

```
us[ername] <name>
```

3.2.1.8 dumpu[sers]

This command prints current content of the USM User table. Format is:

```
# “: ” <name> “;”<engineID> “;” <authProtocol> “;” <privProtocol> “;”
```

where <name> is username, <engineID> is localEngineID, <authProtocol> could be “MD5” or “SHA” and <privProtocol> could only be “DES”.

3.2.1.9 addu[ser]

This command adds new USM user. Syntax is:

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

```
addu[ser] <name> [<authProt><authPswd>[<privProt><privPswd>]]
```

The only mandatory argument is <name>. If other arguments are not supplied user with no authProtocol and no privProtocol will be created. If authProt is present, authPswd must be present as well. The same is true for privacy parameters.

3.2.1.10 modu[ser]

This command modifies existing USM user. Syntax i:

```
modu[ser] <name> [<authProt><authPswd>[<privProt><privPswd>]]
```

Arguments are the same as for adduser.

3.2.1.11 delu[ser]

This command removes existing USM user. Syntax is:

```
delu[ser] <name>
```

3.2.1.12 l[oad]

Without parameters this command prints currently loaded MIB module names. If parameter is present specified MIB file will be loaded. Syntax is:

```
l[oad] <mib-file-name>
```

3.2.1.13 un[load]

Parameter is MIB module name (not MIB file name). Syntax is:

```
un[load] <mib-module-name>
```

3.2.1.14 g[et]

This command sends SNMP get request. Syntax is:

```
g[et] <oid> [<oid>...]
```

where <oid> can be in either raw (1.3.6.1.4.1.4761.2.1.3.0) or label.index format (tvModel.0).

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

3.2.1.15 sys[descr]

This command is shortcut for "get sysDescr.0"

3.2.1.16 n[ext]

This command sends SNMP geNext request. Syntax is:

```
g[et] <oid> [<oid>...]
```

where <oid> can be in either raw (1.3.6.1.4.1.4761.2.1.3.0) or label.index format (tvModel.0).

3.2.1.17 s[et]

This command sends SNMP set request. Syntax is:

```
g[et] <oid> <value>
```

where <oid> can be in either raw (1.3.6.1.4.1.4761.2.1.3.0) or label.index format (tvModel.0) and <value> is the value that should be applied to specified instance. Note that there is no parameter that specifies syntax for the value. This information is derived from loaded MIBs. If no corresponding LEAF object is found an error is returned.

3.2.1.18 w[alk]

This command implement MIB walk. The only parameter is number of objects to retrieve before stop. If parameter is not present walk will perform GetNext till the end of target MIB.

While walk is in progress, invoking "walk" again will stop it.

3.2.1.19 disc[overEngID]

This command performs SNMPv3 engineID discovery. It sends get request to the agent, specifying user="initial", authoritative engineID is zero length octet string and varbinbd list is empty. SMPv3 agent should respond with report UNKNOWN ENGINE ID, which will be used by local SNMP engine to establish mapping between target address and engine id.

3.2.1.20 syn[chronize]

This command performs SNMPv3 time synchronization for authenticated users. For current user this command sends authenticated request but sets both boots and time to zero. Varbinb list is empty. SMPv3 agent should respond with report NOT IN TIME WINDOW, which will be used by local SNMP engine to synchronize its notion of remote engine's boot and time values.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

3.2.1.21 hex

This command enables (hex 1) or disables (hex 0) printing of raw packets (and some extra values like keys, initial vectors, …). By default hex is off.

3.2.1.22 q[uit]

This command shuts down the agent and exits the application.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

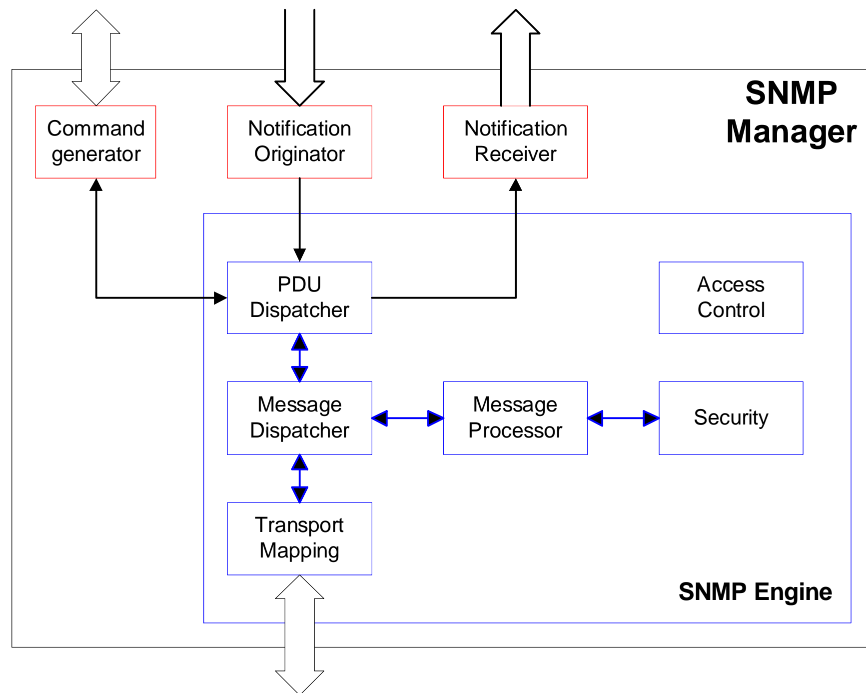
4. Overview of SNMP Architecture

The SNMP architecture, as described in RFC 3411, consists of a distributed, interacting collection of SNMP entities. Each entity may act as an **agent** node, a **manager** node, or a **combination of the two**.

4.1 SNMP Entities

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

4.1.1 SNMP Manager



The SNMP manager interacts with SNMP agents by issuing commands (get, getnext, getbulk, set) and by receiving notification (trap, inform) message. The manager may also interact with other managers by issuing Inform Request PDUs, which provide alerts, and by receiving Inform Response PDUs, which acknowledge Inform Requests. In SNMPv3 terminology, a SNMP manager includes three categories of applications:

- The **Command Generator Applications** monitor and manipulate management data at remote agents. They make use of SNMPv1 and/or SNMPv2 PDUs, including Get, GetNext, GetBulk, and Set.

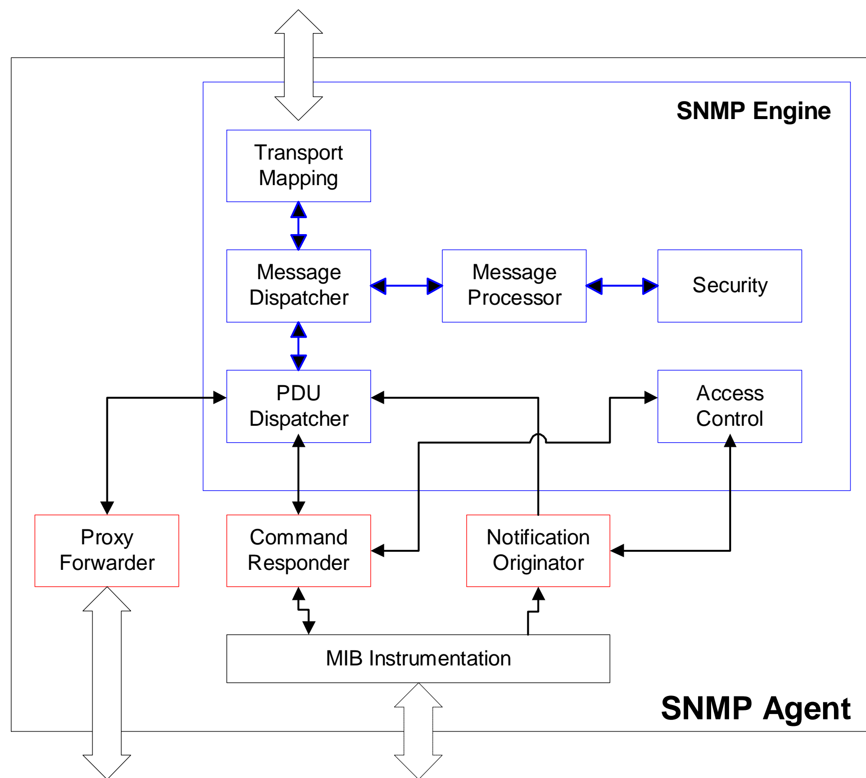
This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

- A **Notification Originator Application** initiates asynchronous messages; in the case of a manager, the InformRequest PDU is used for this application.

- A **Notification Receiver Application** processes incoming asynchronous messages; these include InformRequest, SNMPv2-Trap, and SNMPv1 Trap PDUs. In the case of an incoming InformRequest PDU, the Notification Receiver Application will respond with a Response PDU.

All of the applications just described make use of the services provided by the SNMP engine for this entity.

4.1.2 SNMP Agent



The agent may contain three types of applications:

- A **Command Responder Applications** provide access to management data. These applications respond to incoming requests by retrieving and/or setting managed objects and then issuing a Response PDU.

- A **Notification Originator Application** initiates asynchronous messages; in the case of a agent, the SNMPv2-Trap or SNMPv1 Trap PDU is used for this application.

- A **Proxy Forwarder Application** forwards messages between entities.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

The SNMP engine for a agent has all of the components found in the SNMP engine for a manager, plus an **Access Control Subsystem**. This subsystem provides authorization services to control access to MIBs for the reading and setting of management objects. These services are performed on the basis of the contents of PDUs. An implementation of the Security Subsystem may support one or more distinct access control models. So far, the only defined security model is the View-Based Access Control Model (VACM) for SNMPv3, specified in RFC 3415.

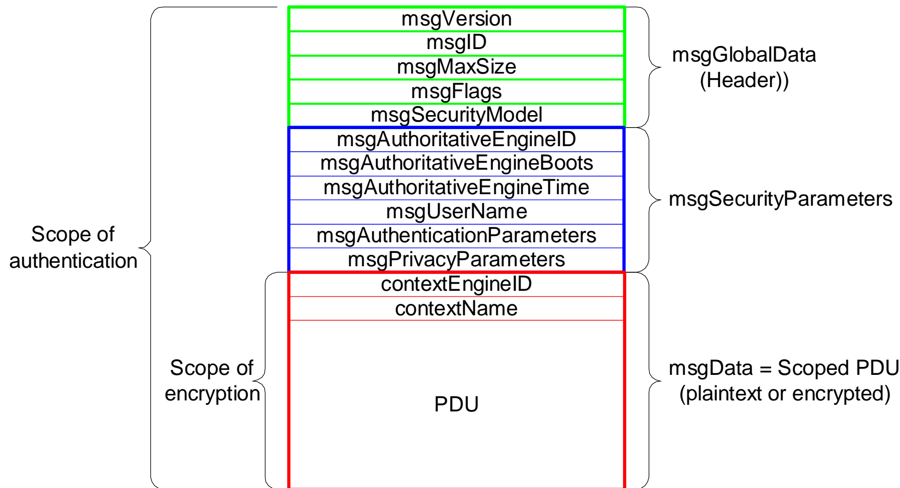
4.2 SNMPv3 Applications

The following types of applications have been defined:

- **Command Generator Application --** initiates SNMP Get, GetNext, GetBulk, and/or Set requests, as well as processing the response to a request which it generated.
- **Command Responder Applications –** receives SNMP Get, GetNext, GetBulk, and/or Set requests destined for the local system as indicated by the fact that the contextEngineID in the received request is equal to that of the local engine through which the request was received. The command responder application will perform the appropriate protocol operation, using access control, and will generate a response message to be sent to the request's originator.
- **Notification Generator Application --** conceptually monitors a system for particular events or conditions, and generates Trap and/or Inform messages based on these events or conditions.
- **Notification Receiver Application --** listens for notification messages, and generates response messages when a message containing an Inform PDU is received.
- **Proxy Forwarder Application --** forwards SNMP messages. Implementation of a proxy forwarder application is optional.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

4.2.1 SNMPv3 Message



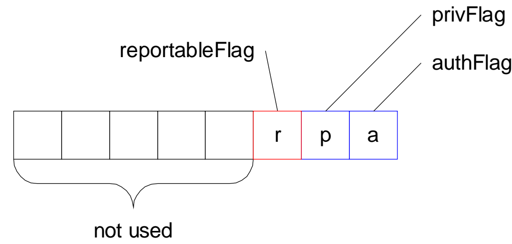
The first five fields (message header) are generated by the message processing model on outgoing messages and processed by the message processing model on incoming messages.

The next six fields show **security parameters** used by USM. Finally, the PDU, together with the contextEngineID and contextName constitute a **scoped PDU**, used for PDU processing.

The message header consists of:

- **msgVersion:** Set to snmpv3.
- **msgID:** A unique identifier used between two SNMP entities to coordinate request and response messages, and by the message processor to coordinate the processing of the message by different subsystem models within the architecture. The range of this ID is 0 through 2^{31}-1.
- **msgMaxSize:** Conveys the maximum size of a message in octets supported by the sender of the message, with a range of 484 through 2^{31}-1. This is the maximum segment size that the sender can accept from another SNMP engine (whether a response or some other message type).
- **msgFlags:** An octet string containing three flags in the least significant three bits: reportableFlag, privFlag, authFlag.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.



If **reportableFlag** = 1, then a Report PDU must be returned to the sender under those conditions that can cause the generation of a Report PDU; when the flag is zero, a Report PDU may not be sent. The reportableFlag is set to 1 by the sender in all messages containing a request (Get, Set) or an Inform, and set to 0 for messages containing a Response, a Trap, or a Report PDU.

The reportableFlag is a secondary aid in determining when to send a Report. It is only used in cases in which the PDU portion of the message cannot be decoded (e.g., when decryption fails due to incorrect key).

The **privFlag** and **authFlag** are set by the sender to indicate the security level that was applied to the message. Following combinations are allowed:

privFlag	authFlag	
0	0	No security applied
0	1	Authenticated message
1	0	INVALID COMBINATION (encryption without authentication is not allowed)
1	1	Authenticated and encrypted message

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

Value	Security model
1	SNMPv1
2	SNMPv2c
3	USM (user based security)

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

4.3 Authentication, Privacy Services and Access Control

4.3.1 User-Based Security Model

4.3.1.1 Overview

RFC 3414 defines the User Security Model (USM). The USM provides authentication and privacy services for SNMP and it is designed to secure against the following threats:

- **Modification of Information:** An entity could alter an in-transit message generated by an authorized entity in such a way as to effect unauthorized management operations, including the setting of object values.
- **Masquerade:** Management operations that are not authorized for some entity may be attempted by that entity by assuming the identity of an authorized entity.
- **Message Stream Modification:** SNMP is designed to operate over a connectionless transport protocol. There is a threat that SNMP messages could be reordered, delayed, or replayed (duplicated) to effect unauthorized management operations.
- **Disclosure:** An entity could observe exchanges between a manager and an agent and thereby learn the values of managed objects and learn of trap events.

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

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

4.3.1.2 Cryptographic Functions

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

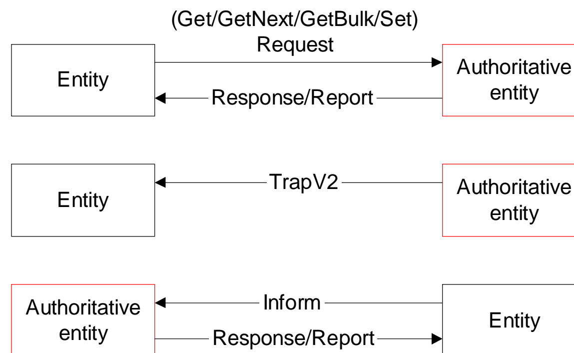
- **Local users:** Any principal at this SNMP engine for which management operations are authorized.
- **Remote users:** Any principal at a remote SNMP engine for which communication is desired.

The values of privacy key and authentication key are not accessible via SNMP. USM allows the use of one of two alternative authentication protocols: HMAC-MD5-96 and HMAC-SHA-96. For encryption USM uses the cipher block chaining (CBC) mode of the Data Encryption Standard (DES).

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

4.3.1.3 Authoritative and Non-Authoritative Engines

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



4.3.1.4 USM Message Parameters

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

- **msgAuthoritativeEngineID:** The snmpEngineID of the authoritative SNMP engine involved in the exchange of this message. Thus, this value refers to the source for a Trap, Response, or Report, and to the destination for a Get, GetNext, GetBulk, Set, or Inform.

- **msgAuthoritativeEngineBoots:** The snmpEngineBoots value of the authoritative SNMP engine involved in the exchange of this message. The object snmpEngineBoots is an integer in the range 0 through 2^{31}-1 that represents the number of times that this SNMP engine has initialized or reinitialized itself since its initial configuration.

- **msgAuthoritativeEngineTime:** The snmpEngineTime value of the authoritative SNMP engine involved in the exchange of this message. The object snmpEngineTime is an integer in the range 0 through 2^{31}-1 that represents the number of seconds since this authoritative SNMP engine last incremented the snmpEngineBoots object. Each authoritative SNMP engine is responsible for incrementing its own snmpEngineTime value once per second. A non-authoritative engine is responsible for incrementing its notion of snmpEngineTime for each remote authoritative engine with which it communicates.

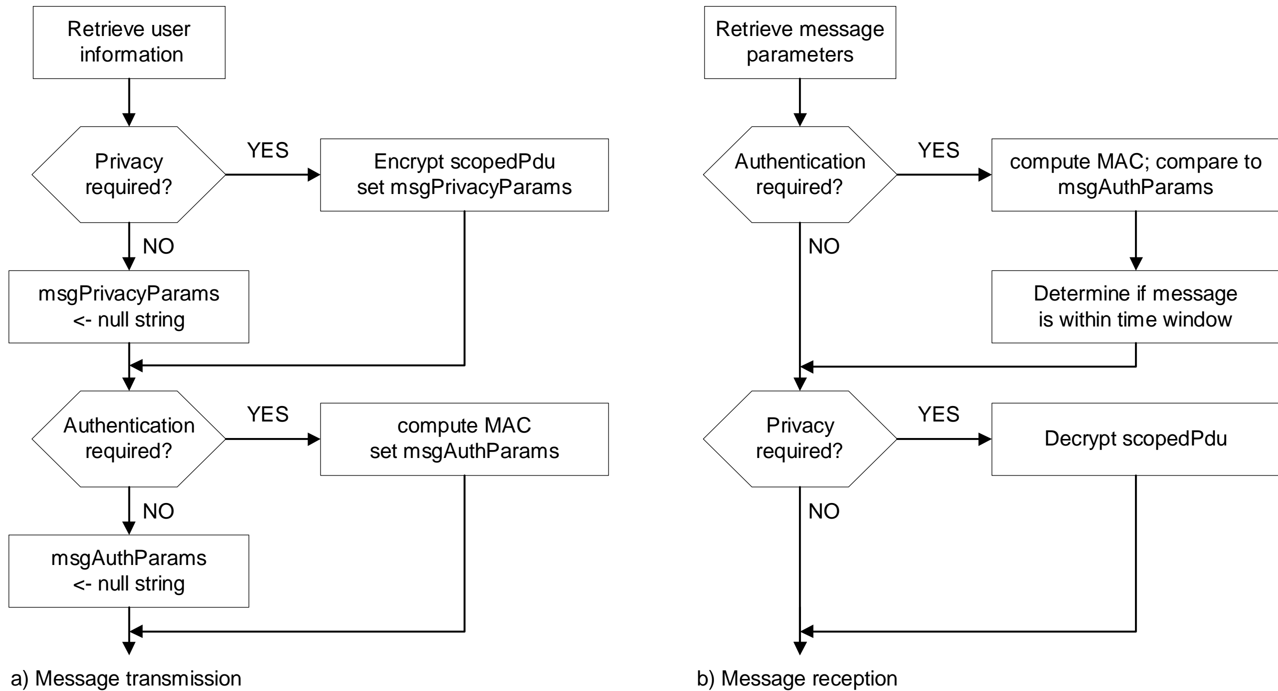
- **msgUserName:** The user (principal) on whose behalf the message is being exchanged.

- **msgAuthenticationParameters:** Null if authentication is not being used for this exchange. Otherwise, this is an authentication parameter. For the current definition of USM, the authentication parameter is an HMAC message authentication code.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

- **msgPrivacyParameters:** Null if privacy is not being used for this exchange. Otherwise, this is a privacy parameter. For the current definition of USM, the privacy parameter is a value used to form the initial value (IV) in the DES CBC algorithm.

The following figure summarizes the operation of USM:



4.3.1.5 Timeliness Mechanisms

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

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

snmpEngineBoots: the most recent value of snmpEngineBoots for the remote authoritative engine.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

snmpEngineTime: this engine's notion of snmpEngineTime for the remote authoritative engine. This value is synchronized to the remote authoritative engine by the synchronization process described below. Between synchronization events, this value is logically incremented once per second to maintain a loose synchronization with the remote authoritative engine.

latestReceivedEngineTime: the highest value of msgAuthoritativeEngineTime that has been received by the this engine from the remote authoritative engine; this value is updated whenever a larger value of msgAuthoritativeEngineTime is received. The purpose of this variable is to protect against a replay message attack that would prevent the non-authoritative SNMP engine's notion of snmpEngineTime from advancing.

One set of these three variables is maintained for each remote authoritative engine known to this engine.

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

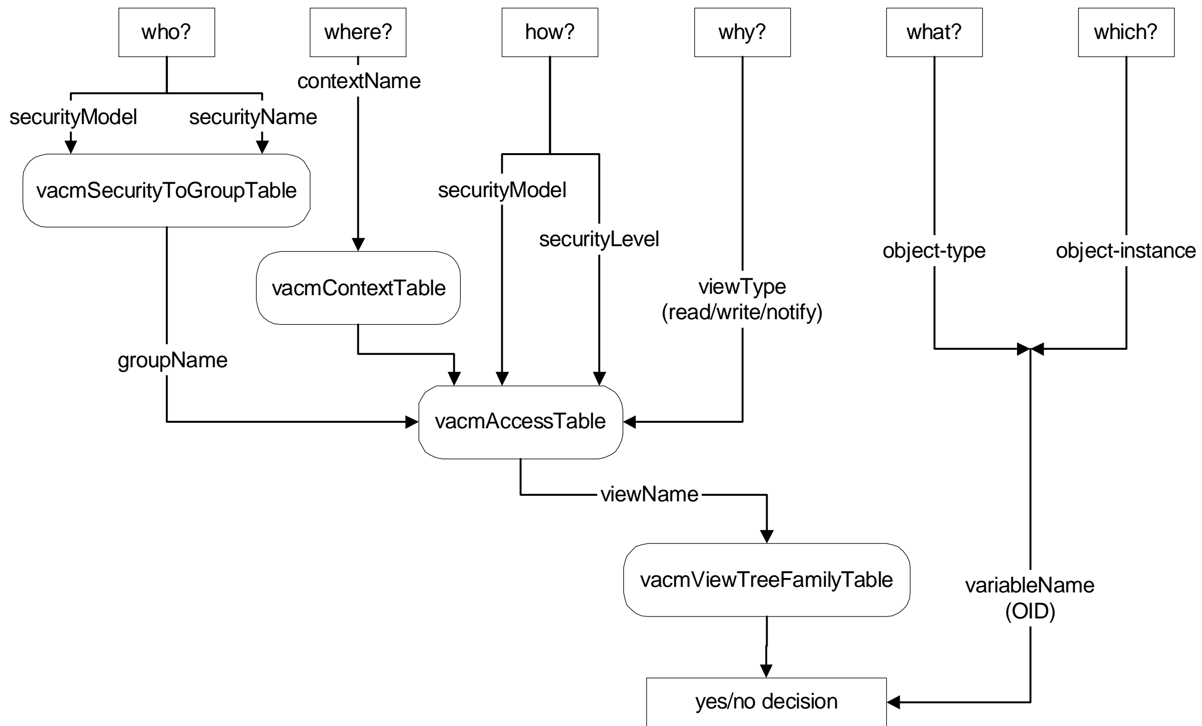
4.3.2 View-Based Access Control

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

VACM makes use of SNMP-VIEW-BASED-ACM-MIB that defines the access control policy for this agent and makes it possible for remote configuration to be used.

Following figure (RFC 3415), shows how the various tables in the VACM MIB come into play in making the access control decision.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.



who: The combination of securityModel and securityName define the who of this operation; it identifies a given principal whose communications are protected by a given securityModel. This combination belongs to at most one group at this SNMP engine.

The vacmSecurityToGroupTable provides the groupName, given the securityModel and securityName.

where: The contextName specifies where the desired management object is to be found. The vacmContextTable contains a list of the recognized contextNames.

how: The combination of securityModel and securityLevel defines how the incoming request or Inform PDU was protected. The combination of who, where, and how identifies zero or one entries in vacmAccessTable.

why: The viewType specifies why access is requested: for a read, write, or notify operation. The selected entry in vacmAccessTable contains one MIB viewName for each of these three types of operation, and viewType is used to select a specific viewName. This viewName selects the appropriate MIB view from vacmViewTreeFamilyTable.

what: The variableName is an object identifier whose prefix identifies a specific object type and whose suffix identifies a specific object instance. The object type indicates what type of management information is requested.

which: The object instance indicates which specific item of information is requested.

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

5. References

------------v1------------

[RFC1155] Rose, M. and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP - based internets", STD 16, RFC 1155, May 1990.

[RFC1157] Case, J., M. Fedor, M. Schoffstall and J. Davin, "The Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

------------v2------------

[RFC1901] The SNMPv2 Working Group, Case, J., McCloghrie, K.,Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[RFC2578] Structure of Management Information Version 2 (SMIv2) K. McCloghrie, D. Perkins, J. Schoenwaelder, April 1999. IETF Standard #58 STANDARD (Obsoletes RFC1902), txt=87K

[RFC2579] Textual Conventions for SMIv2. K. McCloghrie, D. Perkins, J. Schoenwaelder, April 1999. IETF Standard #58 STANDARD (Obsoletes RFC1903), txt=57K

[RFC2580] Conformance Statements for SMIv2. K. McCloghrie, D. Perkins, J. Schoenwaelder, April 1999. IETF Standard #58 STANDARD (Obsoletes RFC1904), txt=52K

[RFC3416] Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). R. Presuhn, Ed.. December 2002. (Format: TXT=70043 bytes) (Obsoletes RFC1905) (Also STD0062) (Status: STANDARD)

[RFC3417] Transport Mappings for the Simple Network Management Protocol (SNMP). R. Presuhn, Ed.. December 2002. (Format: TXT=38650 bytes) (Obsoletes RFC1906) (Also STD0062) (Status: STANDARD)

[RFC3418] Management Information Base (MIB) for the Simple Network Management Protocol (SNMP). R. Presuhn, Ed.. December 2002. (Format: TXT=49096 bytes) (Obsoletes RFC1907) (Also STD0062) (Status: STANDARD)

------------v3------------

[RFC3411] An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. D. Harrington, R. Presuhn, B. Wijnen. December 2002. (Format: TXT=140096 bytes) (Obsoletes RFC2571) (Also STD0062) (Status: STANDARD)

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.

[RFC3412] Message Processing and Dispatching for the Simple Network Management Protocol (SNMP). J. Case, D. Harrington, R. Presuhn, B. Wijnen. December 2002. (Format: TXT=95710 bytes) (Obsoletes RFC2572) (Also STD0062) (Status: STANDARD)

[RFC3413] Simple Network Management Protocol (SNMP) Applications. D. Levi, P. Meyer, B. Stewart. December 2002. (Format: TXT=153719 bytes) (Obsoletes RFC2573) (Also STD0062) (Status: STANDARD)

[RFC3414] User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). U. Blumenthal, B. Wijnen. December 2002. (Format: TXT=193558 bytes) (Obsoletes RFC2574) (Also STD0062) (Status: STANDARD)

[RFC3415] View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). B. Wijnen, R. Presuhn, K. McCloghrie. December 2002. (Format: TXT=82046 bytes) (Obsoletes RFC2575) (Also STD0062) (Status: STANDARD)

This document and associated demo software are provided for informational purposes only, 'as is' and without warranty of any kind.
This document may be copied, however without any modification, and all pages and notices must be included.