Network Working Group
Internet-Draft
Expires: November 4, 2006

D. Harrington
Futurewei Technologies
J. Schoenwaelder
International University Bremen
May 3, 2006

Transport Mapping Security Model (TMSM) Architectural Extension for the Simple Network Management Protocol (SNMP)

draft-ietf-isms-tmsm-02.txt

Status of This Memo

Copyright Notice

Abstract

This document describes a Transport Mapping Security Model (TMSM) extension for the Simple Network Management Protocol (SNMP) architecture defined in RFC 3411. This document identifies and discusses some key aspects that need to be considered for any transport-mapping-based security model for SNMP.

This memo also defines a portion of the Management Information Base (MIB) for managing sessions in the Transport Mapping Security Model extension.

Table of Contents

1. Introduction

This document describes a Transport Mapping Security Model (TMSM) extension for the Simple Network Management Protocol (SNMP) architecture defined in [RFC3411]. This document identifies and discusses some key aspects that need to be considered for any transport-mapping-based security model for SNMP.

1.1. The Internet-Standard Management Framework

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of RFC 3410 [RFC3410].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB module that is compliant to the SMIv2, which is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

1.2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Some points requiring further WG research and discussion are identified by [discuss] markers in the text. Some points requiring further editing by the editors are marked [todo] in the text.

1.3. Acronyms

This section contains a list of acronyms used with the document and references to where in the document the acronym is defined, for easy lookup.

- TMSM - a Transport Mapping Security Model
- MPSP - s Messaging Processing Security Processor, the portion of a TMSM security model that resides in the Message Processing subsytem of an SNMPv3 engine. See Section 2.2.1
- TMSP - the Transport Mapping Security Processor, the portion of a TMSM security model that resides in the Transport Mapping section of the Dispatcher of an SNMPv3 engine. See Section 2.2.1

1.4. Motivation

There are multiple ways to secure one's home or business, but they largely boil down to a continuum of alternatives. Let's consider three general approaches. In the first approach, an individual could buy a gun, learn to use it, and sit on your front porch waiting for intruders. In the second approach, one could hire an employee with a gun, schedule the employee, position the employee to guard what you want protected, hire a second guard to cover if the first gets sick, and so on. In the third approach, you could hire a security company, tell them what you want protected, and they could hire employees, train them, buy the guns, position the guards, schedule the guards, send a replacement when a guard cannot make it, etc., thus providing the security you want, with no significant effort on your part other than identifying requirements and verifying the quality of the service being provided.

The User-based Security Model (USM) as defined in [RFC3414] largely uses the first approach - it provides its own security. It utilizes existing mechanisms (MD5=the gun), but provides all the coordination. USM provides for the authentication of a principal, message encryption, data integrity checking, timeliness checking, etc.

USM was designed to be independent of other existing security infrastructures. USM therefore requires a separate user and key management infrastructure. Operators have reported that deploying another user and key management infrastructure in order to use SNMPv3 is a deterrent to deploying SNMPv3. It is possible but difficult to define external mechanisms that handle the distribution of keys for use by the USM approach.

A solution based on the second approach might use a USM-compliant architecture, but combine the authentication mechanism with an external mechanism, such as RADIUS [RFC2865], to provide the authentication service. It might be possible to utilize an external protocol to encrypt a message, to check timeliness, to check data integrity, etc. It is difficult to cobble together a number of subcontracted services and coordinate them however, because it is difficult to build solid security bindings between the various services, and potential for gaps in the security is significant.

A solution based on the third approach might utilize one or more lower-layer security mechanisms to provide the message-oriented security services required. These would include authentication of the sender, encryption, timeliness checking, and data integrity checking. There are a number of IETF standards available or in development to address these problems through security layers at the transport layer or application layer, among them TLS [RFC4366], SASL

[RFC2222], and SSH [RFC4251].

From an operational perspective, it is highly desirable to use security mechanisms that can unify the administrative security management for SNMPv3, command line interfaces (CLIs) and other management interfaces. The use of security services provided by lower layers is the approach commonly used for the CLI, and is also the approach being proposed for NETCONF [I-D.ietf-netconf-ssh].

This document proposes a Transport Mapping Security Model (TMSM) extension to the RFC3411 architecture, that allows security to be provided by an external protocol connected to the SNMP engine through an SNMP transport-mapping [RFC3417]. Such a TMSM would then enable the use of existing security mechanisms such as (TLS) [RFC4366] or SSH [RFC4251] within the RFC3411 architecture.

There are a number of Internet security protocols and mechanisms that are in wide spread use. Many of them try to provide a generic infrastructure to be used by many different application layer protocols. The motivation behind TMSM is to leverage these protocols where it seems useful.

There are a number of challenges to be addressed to map the security provided by a secure transport into the SNMP architecture so that SNMP continues to work without any surprises. These challenges are discussed in detail in this document. For some key issues, design choices are discussed that may be made to provide a workable solution that meets operational requirements and fits into the SNMP architecture defined in [RFC3411].

2. Requirements of a Transport Mapping Security Model

2.1. Message Security Requirements

Transport mapping security protocols SHOULD ideally provide the protection against the following message-oriented threats [RFC3411]:

1. modification of information
2. masquerade
3. message stream modification
4. disclosure

According to [RFC3411], it is not required to protect against denial of service or traffic analysis.

2.1.1. Security Protocol Requirements

There are a number of standard protocols that could be proposed as possible solutions within the TMSM framework. Some factors should be considered when selecting a protocol for use within this framework.

Using a protocol in a manner for which it was not designed has numerous problems. The advertised security characteristics of a protocol may depend on its being used as designed; when used in other ways, it may not deliver the expected security characteristics. It is recommended that any proposed model include a discussion of the applicability statement of the protocols to be used.

A protocol used for the TMSM framework should ideally require no modifications to the protocol. Modifying the protocol may change its security characteristics in ways that would impact other existing usages. If a change is necessary, the change should be an extension that has no impact on the existing usages. It is recommended that any proposed model include a discussion of potential impact on other usages of the protocol.

It has been a long-standing requirement that SNMP be able to work when the network is unstable, to enable network troubleshooting and repair. The UDP approach has been considered to meet that need well, with an assumption that getting small messages through, even if out of order, is better than getting no messages through. There has been a long debate about whether UDP actually offers better support than TCP when the underlying IP or lower layers are unstable. There has been recent discussion of whether operators actually use SNMP to troubleshoot and repair unstable networks.

There has been discussion of ways SNMP could be extended to better support management/monitoring needs when a network is running just fine. Use of a TCP transport, for example, could enable larger message sizes and more efficient table retrievals.

TMSM models MUST be able to coexist with other protocol models, and may be designed to utilize either TCP or UDP, depending on the transport.

2.2. SNMP Requirements

2.2.1. Architectural Modularity Requirements

SNMP version 3 (SNMPv3) is based on a modular architecture (described in [RFC3411] section 3) to allow the evolution of the SNMP protocol standards over time, and to minimize side effects between subsystems when changes are made. The architecture includes a Security

Subsystem which is responsible for realizing security services.

In SNMPv2, there were many problems of side effects between subsystems caused by the manipulation of MIB objects, especially those related to authentication and authorization, because many of the parameters were stored in shared MIB objects, and different models and protocols could assign different values to the objects. Contributors assumed slightly different shades of meaning depending on the models and protocols being used. As the shared MIB module design was modified to accommodate a specific model, other models which used the same MIB objects were broken.

Abstract Service Interfaces (ASIs) were developed to pass model-independent parameters. The models were required to translate from their model-dependent formats into a model-independent format, defined using model-independent semantics, which would not impact other models.

Parameters have been provided in the ASIs to pass model-independent information about the authentication that has been provided. These parameters include a model-independent identifier of the security "principal", the security model used to perform the authentication, and which SNMP-specific security features were applied to the message (authentication and/or privacy).

Parameters have been provided in the ASIs to pass model-independent transport address information. These parameters utilize the TransportType and TransportAddress

The design of a transport mapping security model must abide the goals of the RFC3411 architecture defined in [RFC3411]. To that end, this transport mapping security model proposal uses a modular design that can be advanced through the standards process independently of other proposals, and independent of other modular components as much as possible.

IETF standards typically require one mandatory to implement solution, with the capability of adding new security mechanisms in the future. Any transport mapping security model should define one minimum-compliance mechanism, preferably one which is already widely deployed within the transport layer security protocol used.

The TMSM architectural extension permits additional transport security protocols to be "plugged into" the RFC3411 architecture, supported by corresponding transport-security-aware transport mapping models.

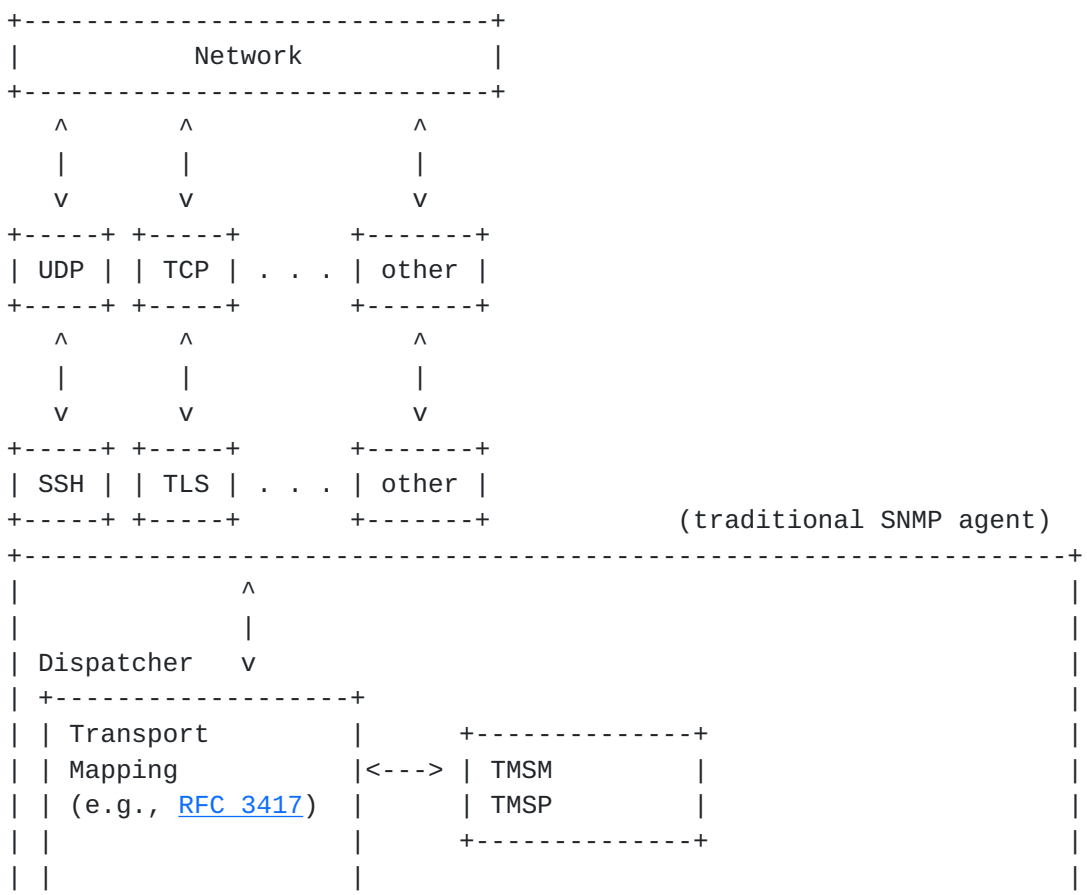
The RFC3411 architecture, and the USM approach, assume that a

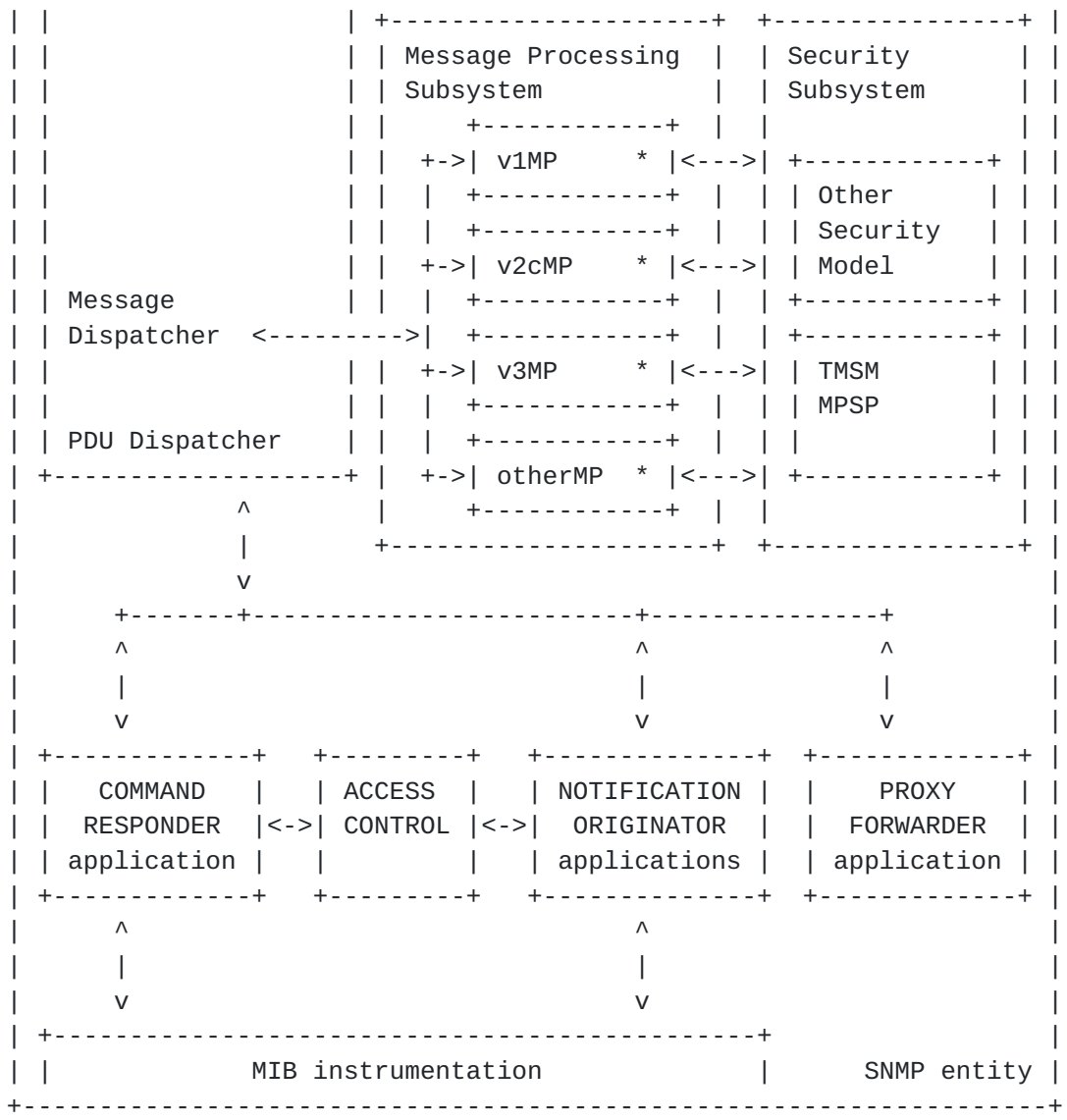
security model is called by a message-processing model and will perform multiple security functions. The TMSM approach performs similar functions but performs them in different places within the architecture, so we need to distinguish the two locations for security processing.

Transport mapping security is by its very nature a security layer which is plugged into the RFC3411 architecture between the transport layer and the message dispatcher. Conceptually, transport mapping security processing will be called from within the Transport Mapping functionality of an SNMP engine dispatcher to perform the translation of transport security parameters to/from security-model-independent parameters. This transport mapping security processor will be referred to in this document as TMSP.

Additional functionality may be performed as part of the message processing function, i.e., in the security subsystem of the RFC3411 architecture. This document will refer to message processor's security processor as the MPSP.

Thus a TMSM is composed of both a TMSP and an MPSP.





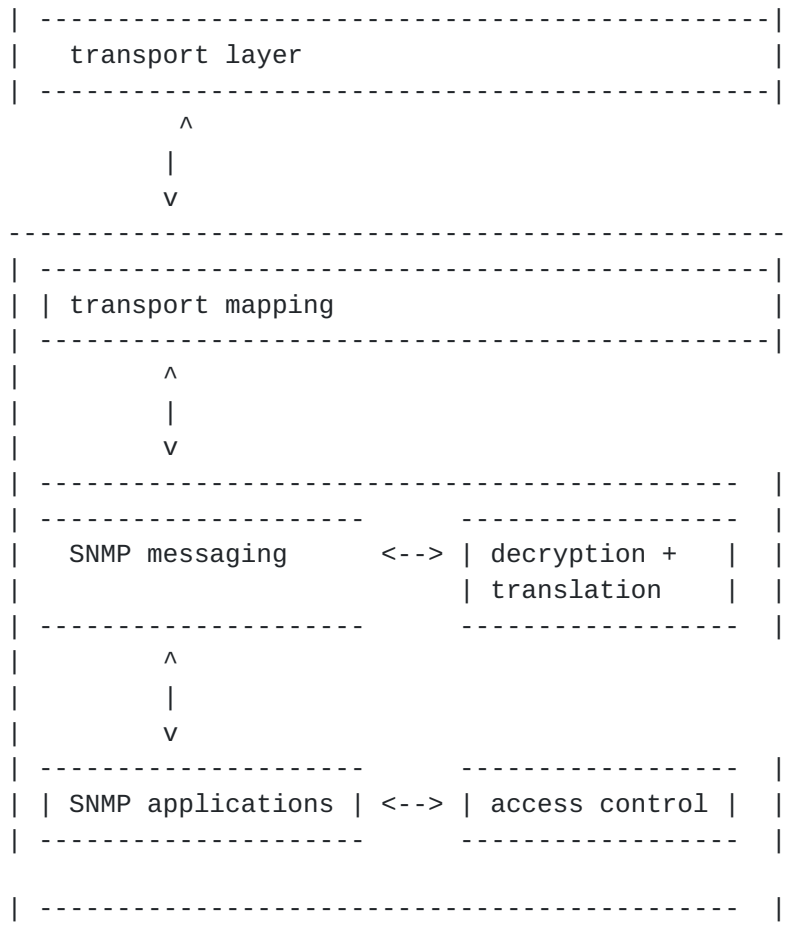
2.2.1.1. USM and the RFC3411 Architecture

The following diagrams illustrate the difference in the security processing done by the USM model and the security processing done by a TMSM model.

The USM security model is encapsulated by the messaging model, because the messaging model needs to perform the following steps (for incoming messages)
1) decode the ASN.1 (messaging model)
2) determine the SNMP security model and parameters (messaging model)

3) decrypt the encrypted portions of the message (security model)
4) translate parameters to model-independent parameters (security model)
5) determine which application should get the decrypted portions (messaging model), and
6) pass on the decrypted portions with model-independent parameters.

The USM approach uses SNMP-specific message security and parameters.

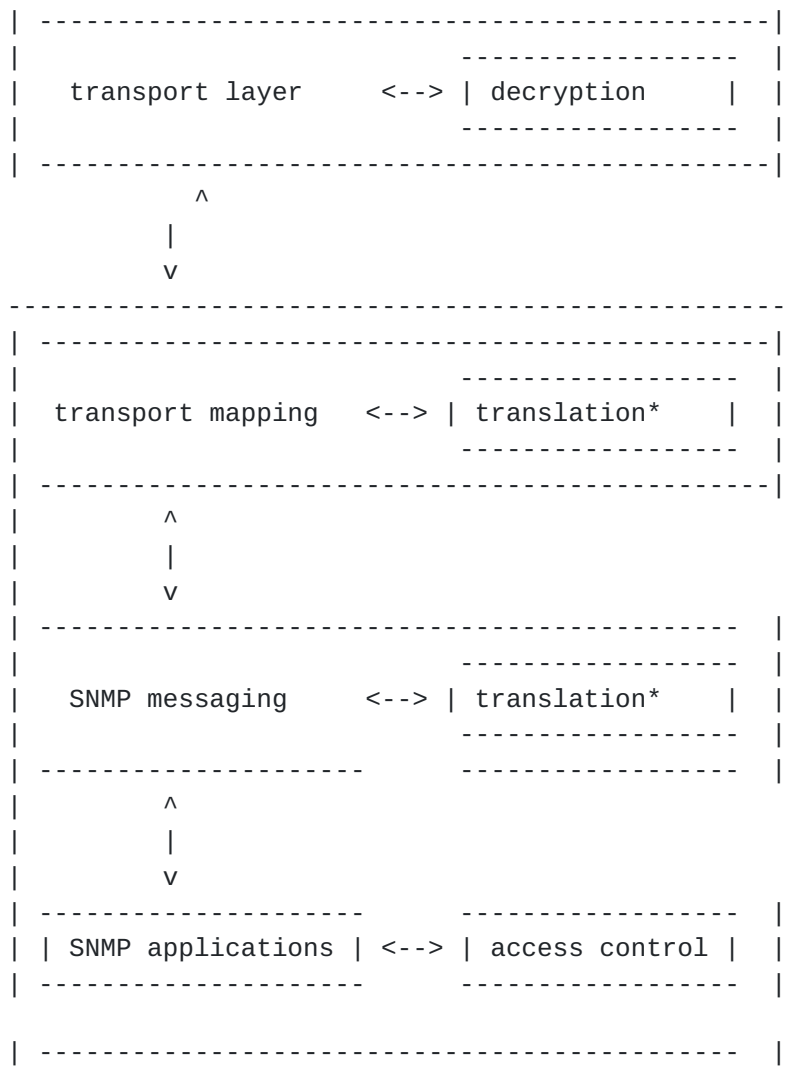


2.2.1.2. TMSM and the RFC3411 Architecture

In the TMSM approach, the order of the steps differ and may be handled by different subsystems:
1) decrypt the encrypted portions of the message (transport layer)

2) determine the SNMP security model and parameters (transport mapping)
3*) translate parameters to model-independent parameters (transport mapping)
4) decode the ASN.1 (messaging model)
5) determine which application should get the decrypted portions (messaging model)
6*) translate parameters to model-independent parameters (security model)
7) pass on the decrypted portions with model-independent security parameters

This is largely based on having non-SNMP-specific message security and parameters. The transport mapping model might provide the translation from e.g., an SSH user name to the securityName in step 3, OR the SSH user might be passed to the messaging model to pass to a TMSM security model to do the translation in step 6, if the WG decides all translations should use the same translation table (e.g., the USM MIB).



2.2.1.3. Passing Information between Engines

A TMSM model will establish an encrypted tunnel between the transport mappings of two SNMP engines. One transport mapping security model instance encrypts all messages, and the other transport mapping security model instance decrypts the messages.

After the transport layer tunnel is established, then SNMP messages can conceptually be sent through the tunnel from one SNMP message dispatcher to another SNMP message dispatcher. Once the tunnel is established, multiple SNMP messages may be able to be passed through the same tunnel.

2.2.2. Access Control Requirements

2.2.2.1. securityName Binding

For SNMP access control to function properly, the security mechanism must establish a securityModel identifier, a securityLevel, and a securityName, which is the security model independent identifier for a principal. The SNMPv3 message processing architecture subsystem relies on a security model, such as USM, to play a role in security that goes beyond protecting the message - it provides a mapping between the USM-specific principal to a security-model independent securityName which can be used for subsequent processing, such as for access control.

The TMSM is a two-stage security model, with a transport mapping security process (TMSP) and a message processing security process (MPSP). Depending on the design of the specific TMSM model, i.e., which transport layer protocol is used, different features might be provided by the TMSP or by the MPSP. For example, the translation from a mechanism-specific authenticated identity to a securityName might be done by the TMSP or by the MPSP.

The securityName MUST be bound to the mechanism-specific authenticated identity, and this mapping MUST be done before the MPSP portion of the model passes securityName to the message processing model via the processIncoming() ASI.

The SNMP architecture distinguishes between messages with no authentication and no privacy (noAuthNoPriv), authentication without privacy (authNoPriv) and authentication with privacy (authPriv). Hence, the authentication of a transport layer identity plays an important role and must be considered by any TMSM, and user authentication must be available via the transport layer security protocol.

If the type of authentication provided by the transport layer (e.g. TLS) is considered adequate to secure and/or encrypt the message, but inadequate to provide the desired granularity of access control (e.g. user-based), then a second authentication (e.g., one provided by a RADIUS server) may be used to provide the authentication identity which is bound to the securityName. This approach would require a good analysis of the potential for man-in-the-middle attacks or masquerade possibilities.

2.2.2.2. Separation of Authentication and Authorization

A TMSM security model should take care to not violate the separation of authentication and authorization in the RFC3411 architecture. The

isAccessAllowed() primitive is used for passing security-model independent parameters between the subsystems of the architecture.

Mapping of (securityModel, securityName) to an access control policy should be handled within the access control subsystem, not the security subsystem, to be consistent with the modularity of the RFC3411 architecture. This separation was a deliberate decision of the SNMPv3 WG, to allow support for authentication protocols which did not provide authorization capabilities, and to support authorization schemes, such as VACM, that do not perform their own authentication.

An authorization model MAY require authentication by certain securityModels and a minimum securityLevel to allow access to the data.

TMSM is an enhancement for the SNMPv3 privacy and authentication provisions, but it is not a significant improvement for the authorization needs of SNMPv3. TMSM provides all the model-independent parameters for the isAccessAllowed() primitive [RFC3411].

TMSM does not specify how the securityModel and securityName could be dynamically mapped to a VACM-style groupName. The mapping of (securityModel, securityName) to a groupName is a VACM-specific mechanism for naming an access control policy, and for tying the named policy to the addressing capabilities of the data modeling language (e.g. SMIv2 [RFC2578]), the operations supported, and other factors. Providing a binding outside the Access Control subsystem might create dependencies that could make it harder to develop alternate models of access control, such as one built on UNIX groups or Windows domains. The preferred approach is to pass the model-independent security parameters via the isAccessAllowed() ASI, and perform the mapping within the access control model.

To provide support for protocols which simultaneously send information for authentication and authorization, such as RADIUS [RFC2865], model-specific authorization information MAY be cached or otherwise made available to the access control subsystem, e.g., via a MIB table similar to the vacmSecurityToGroupTable, so the access control subsystem can create an appropriate binding between the model-independent securityModel and securityName and a model-specific access control policy. This may be highly undesirable, however, if it creates a dependency between a security model and an access control model, just as it is undesirable that the TMSM approach creates a dependency between a TMSP and an MPSP.

2.2.3. Security Parameter Passing Requirements

RFC3411 section 4 describes primitives to describe the abstract data flows between the various subsystems, models and applications within the architecture. Abstract Service Interfaces describe the flow of data between subsystems within an engine. The ASIs generally pass model-independent information.

Within an engine using a TMSM-based security model, outgoing SNMP messages are passed unencrypted from the message dispatcher to the transport mapping, and incoming messages are passed unencrypted from the transport mapping to the message dispatcher.

The security parameters include a model-independent identifier of the security "principal", the security model used to perform the authentication, and which SNMP-specific security services were (should be) applied to the message (authentication and/or privacy).

In the RFC3411 architecture, which reflects the USM security model design, the messaging model must unpack SNMP-specific security parameters from an incoming message before calling a specific security model to authenticate and decrypt an incoming message, perform integrity checking, and translate model-specific security parameters into model-independent parameters.

In the TMSM approach, the security-model specific parameters are not carried in the SNMP message. The parameters are provided by SNMP applications for outgoing messages, and the parameters for incoming messages are extracted from the transport layer by the security-model-specific transport mapping before the message is passed to the message processing subsystem.

For outgoing messages, it is necessary to have an MPSP because it is the MPSP that actually creates the message from its component parts. Whether there are any security services provided by the MPSP for an outgoing message is model-dependent.

For incoming messages, there might be security functionality that can only be handled after the message version is known. The message version is determined by the Message Processing model and passed to the MPSP via the processIncoming() ASI.

The RFC3411 architecture has no ASI parameters for passing security information between the transport mapping and the dispatcher, and between the dispatcher and the message processing model. If there is a need to have an MPSP called from the message processing model to, for example, verify that msgFlags and the transport security are consistent, then it will be necessary to pass the model-dependent

security parameters from the TMSP through to the MPSP.

This document describes a cache, into which the TMSP puts information about the security applied to an incoming message, and an MPSP extracts that information from the cache. Given that there may be multiple TM-security caches, a tmStateReference is passed as an extra parameter in the ASIs between the transport mapping and the messaging security model.so the MPSP knows which cache of information to consult.

This approach does create dependencies between a model-specific TMSP and a corresponding specific MPSP. This approach of passing a model-independent reference is consistent with the securityStateReference cache already being passed around in the RFC3411 ASIs.

2.3. Session Requirements

Throughout this document, the term session is used. Some underlying secure transports will have a notion of session. Some underlying secure transports might enable the use of channels or other session-like thing. In this document the term session refers to an association between two SNMP engines that permits the secure transmission of one or more SNMP messages within the lifetime of the session. How the session is actually established, opened, closed, or maintained is specific to a particular security model.

Sessions are not part of the SNMP architecture described in [RFC3411], but are considered desirable because the cost of authentication can be amortized over potentially many transactions.

It is important to note that the architecture described in [RFC3411] does not include a session selector in the Abstract Service Interfaces, and neither is that done for this architectural extension, so an SNMP application cannot select the session except by passing a unique combination of securityName, securityModel, and securityLevel.

All TMSM-based security models should discuss the impact of sessions on SNMP usage, including how to establish/open a TMSM session (i.e., how it maps to the concepts of session-like things of the underlying protocol), how to behave when a TMSM session cannot be established, how to close a TMSM session (and the underlying protocol equivalent) properly, how to behave when a TMSM session is closed improperly, the session security properties, session establishment overhead, and session maintenance overhead.

To reduce redundancy, this document will discuss aspects that are expected to be common to all TMSM-based security model sessions.

2.3.1. Session Establishment Requirements

SNMP applications must provide the transport address, securityName, securityModel, and securityLevel to be used for a session.

SNMP Applications typically have no knowledge of whether the session that will be used to carry commands was initially established as a notification session, or a request-response session, and SHOULD NOT make any assumptions based on knowing the direction of the session. If an administrator or security model designer wants to differentiate a session established for different purposes, such as a notification session versus a request-response session, the application can use different securityNames or transport addresses (e.g., port 161 vs port 162) for different purposes.

An SNMP engine containing an application that initiates communication, e.g., a Command Generator or Notification Originator, MUST be able to attempt to establish a session for delivery if a session does not yet exist. If a session cannot be established then the message is discarded.

Sessions are usually established by the transport mapping security processor when no appropriate session is found for an outgoing message, but sessions may be established in advance to support features such as notifications and call-home. How sessions are established in advance is beyond the scope of this document.

Sessions are initiated by notification originators when there is no currently established connection that can be used to send the notification. For a client-server security protocol, this may require provisioning authentication credentials on the agent, either statically or dynamically, so the client/agent can successfully authenticate to a notification receiver.

A TMSM-based security model must be able to determine whether a session does or does not exist, and must be able to determine which session has the appropriate security characteristics (transport address, securityName, securityModel, and securityLevel) for an outgoing message.

A TMSM security model implementation MAY reuse an already established session with the appropriate transport address, securityName, securityModel, and securityLevel characteristics for delivery of a message originated by a different type of application than originally caused the session to be created. For example, an implementation that has an existing session originally established to receive a request may use that session to send an outgoing notification, and may use a session that was originally established to send a

notification to send a request. Responses are expected to be returned using the same session that carried the corresponding request message. Reuse is not required for conformance.

If a session can be reused for a different type of message, but a receiver is not prepared to accept different message types over the same session, then the message MAY be dropped by the manager.

2.3.2. Session Maintenance Requirements

A TMSM-based security model can tear down sessions as needed. It may be necessary for some implementations to tear down sessions as the result of resource constraints, for example.

The decision to tear down a session is implementation-dependent. While it is possible for an implementation to automatically tear down each session once an operation has completed, this is not recommended for anticipated performance reasons. How an implementation determines that an operation has completed, including all potential error paths, is implementation-dependent.

Implementations should be careful to not tear down a session between the time a request is received and the time the response is sent. The elements of procedure for TMSM-based security models should be sure to describe the expected behavior when no session exists for a response.

The elements of procedure may discuss when cached information can be discarded, and the timing of cache cleanup may have security implications, but cache memory management is an implementation issue.

If a security model defines MIB module objects to maintain session state information, then the security model MUST describe what happens to the objects when a related session is torn down, since this will impact interoperability of the MIB module.

2.3.3. Message security versus session security

A TMSM session is associated with state information that is maintained for its lifetime. This state information allows for the application of various security services to TMSM-based security models. Cryptographic keys established at the beginning of the session SHOULD be used to provide authentication, integrity checking, and encryption services for data that is communicated during the session. The cryptographic protocols used to establish keys for a TMSM-based security model session SHOULD ensure that fresh new session keys are generated for each session. If each session uses new session keys, then messages cannot be replayed from one session

to another. In addition sequence information MAY be maintained in the session which can be used to prevent the replay and reordering of messages within a session.

A TMSM session will typically have a single transport address, securityName and securityLevel associated with it. If an exchange between communicating engines would require a different securityLevel or would be on behalf of a different securityName, then another session would be needed. An immediate consequence of this is that implementations should be able to maintain some reasonable number of concurrent sessions.

For TMSM models, securityName is typically specified during session setup, and associated with the session identifier.

SNMPv3 was designed to support multiple levels of security, selectable on a per-message basis by an SNMP application, because there is not much value in using encryption for a Commander Generator to poll for non-sensitive performance data on thousands of interfaces every ten minutes; the encryption adds significant overhead to processing of the messages.

Some TMSM-based security models MAY support only specific authentication and encryption services, such as requiring all messages to be carried using both authentication and encryption, regardless of the security level requested by an SNMP application.

Some security models may use an underlying transport that provides a per-message requested level of authentication and encryption services. For example, if a session is created as 'authPriv', then keys for encryption could still be negotiated once at the beginning of the session. But if a message is presented to the session with a security level of authNoPriv, then that message could simply be authenticated and not encrypted within the same transport session. Whether this is possible depends on the security model and the secure transport used.

If the underlying transport layer security was configurable on a per-message basis, a TMSM-based security model could have a security-model-specific MIB module with configurable maxSecurityLevel and a minSecurityLevel objects to identify the range of possible levels. A session's maxSecurityLevel would identify the maximum security it could provide, and a session created with a minSecurityLevel of authPriv would reject an attempt to send an authNoPriv message. The elements of procedure of the security model would need to describe the procedures to enable this determination.

For security models that do not support variable security services in

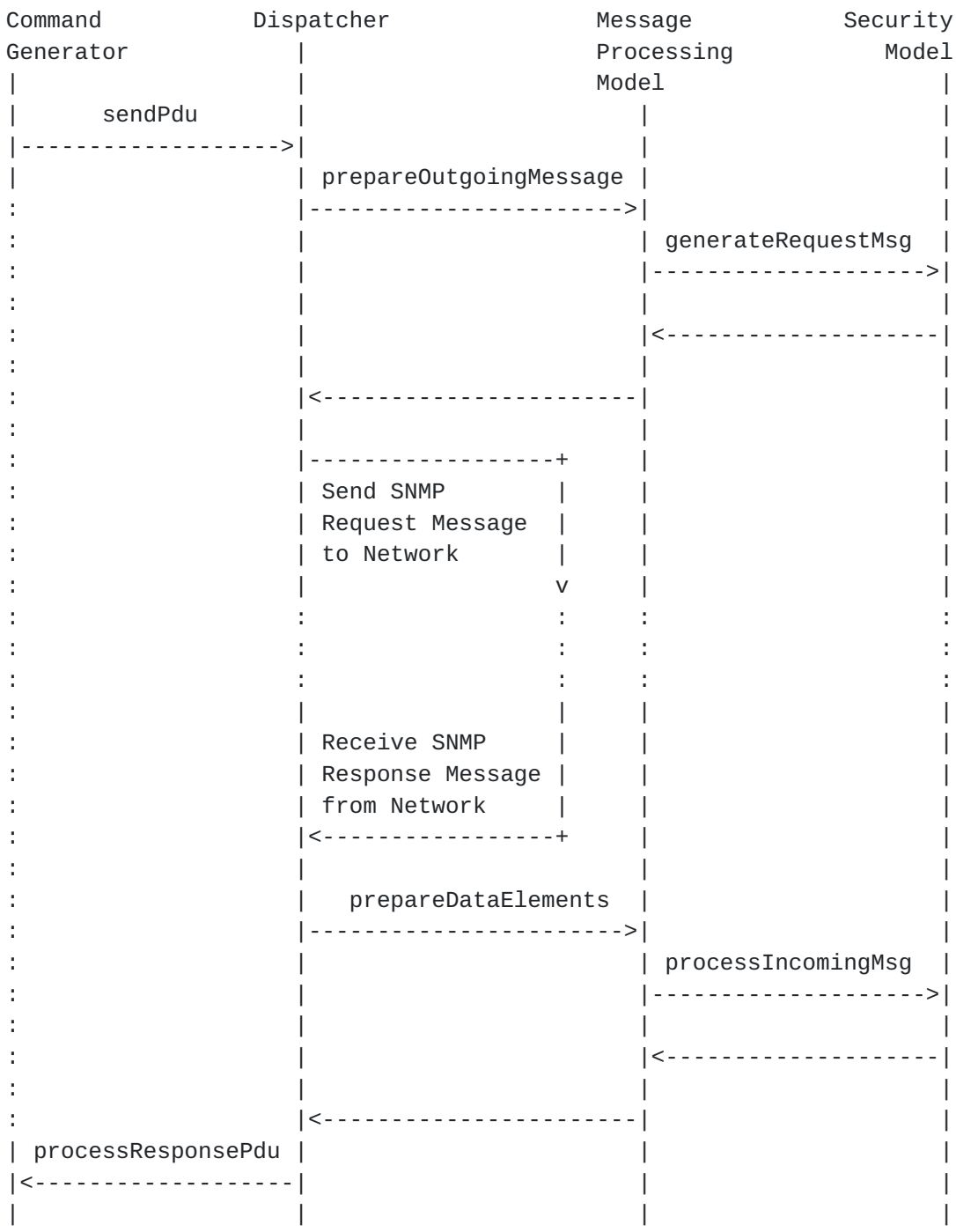
one session, multiple sessions could be established with different security levels, and for every packet the SNMP engine could select the appropriate session based on the requested securityLevel. Some SNMP entities are resource-constrained. Adding sessions increases the need for resources, but so does encrypting unnecessarily. Designers of security models should consider the trade offs for resource-constrained devices.

3. Scenario Diagrams for TMSM

RFC3411 section 4.6 provides scenario diagrams to illustrate how an outgoing message is created, and how an incoming message is processed. Both diagrams are incomplete, however. In section 4.6.1, the diagram doesn't show the ASI for sending an SNMP request to the network or receiving an SNMP response message from the network. In section 4.6.2, the diagram doesn't illustrate the interfaces required to receive an SNMP message from the network, or to send an SNMP message to the network.

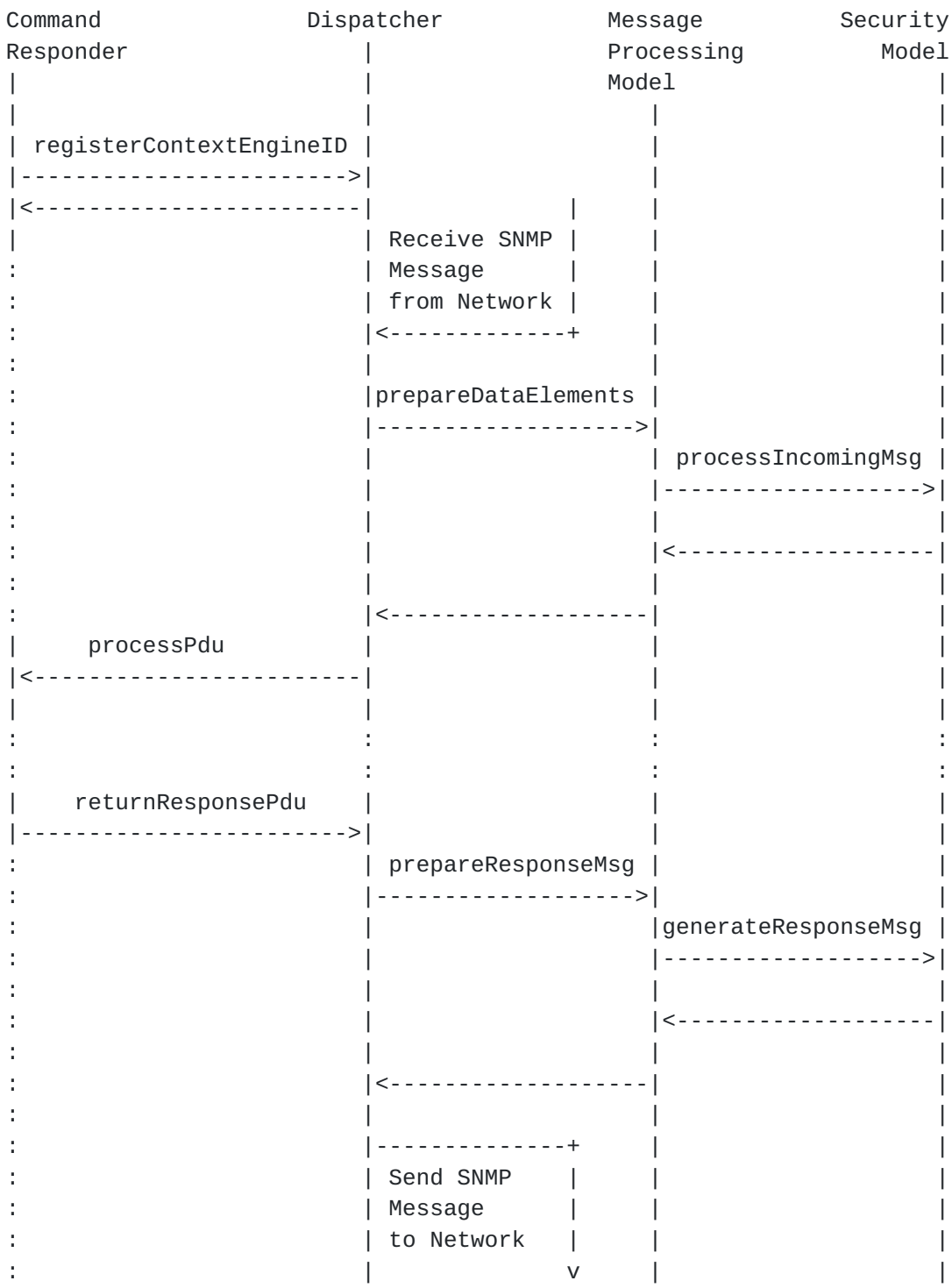
3.1. Command Generator or Notification Originator

This diagram from RFC3411 4.6.1 shows how a Command Generator or Notification Originator application [RFC3413] requests that a PDU be sent, and how the response is returned (asynchronously) to that application.



3.2. Command Responder

This diagram shows how a Command Responder or Notification Receiver application registers for handling a pduType, how a PDU is dispatched to the application after an SNMP message is received, and how the Response is (asynchronously) send back to the network.



4. Abstract Service Interfaces for TMSM

4.1. Existing Abstract Service Interfaces

The OUT parameters of the prepareOutgoingMessage() ASI are used to pass information from the message processing model to the dispatcher and on to the transport mapping:

```
statusInformation =          -- success or errorIndication
prepareOutgoingMessage(
IN  transportDomain          -- transport domain to be used
IN  transportAddress         -- transport address to be used
IN  messageProcessingModel   -- typically, SNMP version
IN  securityModel            -- Security Model to use
IN  securityName             -- on behalf of this principal
IN  securityLevel            -- Level of Security requested
IN  contextEngineID          -- data from/at this entity
IN  contextName              -- data from/in this context
IN  pduVersion               -- the version of the PDU
IN  PDU                      -- SNMP Protocol Data Unit
IN  expectResponse           -- TRUE or FALSE
IN  sendPduHandle            -- the handle for matching
                                incoming responses
OUT  destTransportDomain     -- destination transport domain
OUT  destTransportAddress    -- destination transport address
OUT  outgoingMessage         -- the message to send
OUT  outgoingMessageLength   -- its length
            )
```

4.2. TMSM Abstract Service Interfaces

A set of abstract service interfaces have been defined within this document to describe the conceptual data flows between the Transport Mapping Security Models and adjacent components in the system.

The sendMessage ASI is used to pass a message from the Dispatcher to the transport mapping for sending.

```
statusInformation =
sendMessage(
IN   destTransportDomain          -- transport domain to be used
IN   destTransportAddress         -- transport address to be used
IN   outgoingMessage              -- the message to send
IN   outgoingMessageLength        -- its length
IN   tmStateReference
OUT  sessionID
 )
```

The recvMessage ASI is used to pass a message from the transport mapping to the Dispatcher.

```
statusInformation =
recvMessage(
IN   destTransportDomain           -- transport domain to be used
IN   destTransportAddress          -- transport address to be used
IN   incomingMessage               -- the message received
IN   incomingMessageLength         -- its length
OUT   tmStateReference
OUT   sessionID
 )
```

The Transport Mapping Security Model provides the following primitives to pass data back and forth between the TMSM and specific TMSM-based security models, which provide the interface to the underlying secure transport service. Each TMSM-based security model should define the security-model-specific elements of procedure for the openSession(), closeSession(), txMessage(), and rxMessage() interfaces.

```
statusInformation =
txMessage(
IN   destTransportDomain           -- transport domain to be used
IN   destTransportAddress          -- transport address to be used
IN   outgoingMessage               -- the message to send
IN   outgoingMessageLength         -- its length
IN   tmStateReference
OUT  sessionID
 )

statusInformation =
rxMessage(
IN   destTransportDomain           -- transport domain to be used
IN   destTransportAddress          -- transport address to be used
IN   incomingMessage               -- the message to send
IN   incomingMessageLength         -- its length
OUT   tmStateReference
 )

 statusInformation =
openSession(
IN   transportDomain              -- transport domain to be used
IN   transportAddress             -- transport address to be used
IN   tmStateReference
OUT  sessionID
 )


statusInformation =
closeSession(
IN   sessionID
 )
```

5. Cached Information and References

The RFC3411 architecture uses caches to store dynamic model-specific information, and uses references in the ASIs to indicate in a model-independent manner which cached information must flow between subsytems.

5.1. securityStateReference Cached Security Data

From RFC3411: "For each message received, the Security Model caches the state information such that a Response message can be generated using the same security information, even if the Local Configuration Datastore is altered between the time of the incoming request and the

outgoing response.

A Message Processing Model has the responsibility for explicitly releasing the cached data if such data is no longer needed. To enable this, an abstract securityStateReference data element is passed from the Security Model to the Message Processing Model. The cached security data may be implicitly released via the generation of a response, or explicitly released by using the stateRelease primitive, as described in RFC3411 section 4.5.1."

For the TMSM approach, the TMSP may need to provide the information to be stored in the securityStateReference to the message processing model. such as the security-model-independent securityName, securityLevel, and securityModel parameters. For responses, the messaging model may need to pass the parameters back to the TMSP.

This document will differentiate the tmStateReference provided by the TMSP to the MPSP, from the securityStateReference provided by the MPSP to the Dispatcher. This document does not specify an implementation strategy, only an abstract discussion of the data that must flow between subsystems. An implementation MAY use one cache and one reference to serve both functions, but an implementer must be aware of the cache-release issues to prevent the cache from being released before the transport mapping has had an opportunity to extract the information it needs.

5.2. tmStateReference Cached Security Data

A tmStateReference is used to pass data between the TMSP and the MPSP, similar to the securityStateReference described in RFC3412. A reference to this cache can be envisioned as being appended to the ASIs between the TM and the MP.

The TMSP may provide only some aspects of security, and leave some aspects to the MPSP. tmStateReference should be used to pass any parameters, in a model- and mechanism-specific format, that will be needed to coordinate the activities of the TMSP and MPSP, plus the parameters subsequently passed in securityStateReference. For example, the TMSP may provide privacy and data integrity and authentication and authorization policy retrievals, or some subset of these features, depending on the features available in the transport mechanisms. A field in tmStateReference should identify which services were provided for each received message by the TMSP, the securityLevel applied to the received message, the model-specific security identity, the session identifier for session based transport security, and so on.

6. Integration with the SNMPv3 Message Format

TMSM proposals can use the SNMPv3 message format, defined in RFC3412, section 6. This section discusses how the fields could be reused.

6.1. msgVersion

For proposals that reuse the SNMPv3 message format, this field should contain the value 3.

6.2. msgGlobalData

The fields msgID and msgMaxSize are used identically for the TMSM models as for the USM model.

The msgSecurityModel field should be set to a value from the SnmpSecurityModel enumeration [RFC3411] to identify the specific TMSM model. Each standards-track TMSM model should have an enumeration assigned by IANA. Each enterprise-specific security model should have an enumeration assigned following instructions in the description of the SnmpSecurityModel TEXTUAL-CONVENTION from RFC3411.

The msgSecurityParameters field would carry security information required for message security processing. It is unclear whether this field would be useful or what parameters would be carried to support security, since message security is provided by an external process, and msgSecurityParameters are not used by the access control subsystem.

RFC3412 defines two primitives, generateRequestMsg() and processIncomingMsg() which require the specification of an authoritative SNMP entity. [discuss] We need to discuss what the meaning of authoritative would be in a TMSM environment, whether the specific services provided in USM security from msgSecurityParameters still are needed, and how the Message Processing model provides this information to the security model via generateRequestMsg() and processIncomingMsg() primitives. RFC3412 specifies that "The data in the msgSecurityParameters field is used exclusively by the Security Model, and the contents and format of the data is defined by the Security Model. This OCTET STRING is not interpreted by the v3MP, but is passed to the local implementation of the Security Model indicated by the msgSecurityModel field in the message."

The msgFlags have the same values for the TMSM models as for the USM model. "The authFlag and privFlag fields indicate the securityLevel that was applied to the message before it was sent on the wire."

6.3. securityLevel and msgFlags

For an outgoing message, msgFlags is the requested security for the message; if a TMSM cannot provide the requested securityLevel, the model MUST describe a standard behavior that is followed for that situation. If the TMSM cannot provide at least the requested level of security, the TMSM MUST discard the request and SHOULD notify the message processing model that the request failed.

[discuss] how is yet to be determined, and may be model-specific or implementation-specific.

For an outgoing message, if the TMSM is able to provide stronger than requested security, that may be acceptable. The transport layer protocol would need to indicate to the receiver what security has been applied to the actual message. To avoid the need to mess with the ASN.1 encoding, the SNMPv3 message carries the requested msgFlags, not the actual securityLevel applied to the message. If a message format other than SNMPv3 is used, then the new message may carry the more accurate securityLevel in the SNMP message.

For an incoming message, the receiving TMSM knows what must be done to process the message based on the transport layer mechanisms. If the underlying transport security mechanisms for the receiver cannot provide the matching securityLevel, then the message should follow the standard behaviors for the transport security mechanism, or be discarded silently.

Part of the responsibility of the TMSM is to ensure that the actual security provided by the underlying transport layer security mechanisms is configured to meet or exceed the securityLevel required by the msgFlags in the SNMP message. When the MPSP processes the incoming message, it should compare the msgFlags field to the securityLevel actually provided for the message by the transport layer security. If they differ, the MPSP should determine whether the changed securityLevel is acceptable. If not, it should discard the message. Depending on the model, the MPSP may issue a reportPDU with the XXXXXXX model-specific counter.

7. Prepare an Outgoing SNMP Message

Following RFC3412, section 7.1, the SNMPv3 message processing model uses the generateResponseMsg() or generateRequestMsg() primitives, to call the MPSP. The message processing model, or the MPSP it calls, may need to put information into the tmStateReference cache for use by the TMSP, such as:

tmSecurityStateReference - the unique identifier for the cached information
tmTransportDomain
tmTransportAddress
tmSecurityModel - an indicator of which mechanisms to use
tmSecurityName - a model-specific identifier of the security principal
tmSecurityLevel - an indicator of which security services are requested

A tmStateReference cache may contain additional information such as
tmSessionID
tmSessionKey
tmSessionMsgID

8. Prepare Data Elements from an Incoming SNMP Message

For an incoming message, the TMSP will need to put information from the transport mechanisms used into the tmStateReference so the MPSP can extract the information and add it conceptually to the securityStateReference.

The tmStateReference cache will likely contain at least the following information:
tmStateReference - a unique identifier for the cached information
tmSecurityStateReference - the unique identifier for the cached information
tmTransportDomain
tmTransportAddress
tmSecurityModel - an indicator of which mechanisms to use
tmSecurityName - a model-specific identifier of the security principal
tmSecurityLevel - an indicator of which security services are requested
tmAuthProtocol
tmPrivProtocol

and may contain additional information such as
tmSessionID
tmSessionKey
tmSessionMsgID

9. Notifications

For notifications, if the cache has been released and then session closed, then the MPSP will request the TMSP to establish a session, populate the cache, and pass the securityStateReference to the MPSP.

[discuss] We need to determine what state needs to be saved here.

10. The TMSM MIB Module

This memo defines a portion of the Management Information Base (MIB) for managing sessions in the Transport Mapping Security Model extension.

10.1. Structure of the MIB Module

Objects in this MIB module are arranged into subtrees. Each subtree is organized as a set of related objects. The overall structure and assignment of objects to their subtrees, and the intended purpose of each subtree, is shown below.

10.1.1. The tmsmNotifications Subtree

This subtree contains notifications to alert other entities to events that are applicable to all security models based on the Transport Mapping Security Model extension.

10.1.2. The tmsmStats Subtree

This subtree contains security-model-independent counters which are applicable to all security models based on the .Transport Mapping Security Model extension. This subtree provides information for identifying fault conditions and performance degradation.

10.1.3. The tmsmSession Subtree

This subtree contains security-model-independent information about sessions which are applicable to all security models based on the Transport Mapping Security Model extension.

10.2. Relationship to Other MIB Modules

Some management objects defined in other MIB modules are applicable to an entity implementing this MIB. In particular, it is assumed that an entity implementing the TMSM-MIB module will also implement the SNMPv2-MIB [RFC3418].

This MIB module is expected to be used with the MIB modules defined for managing specific security models that are based on the TMSM extension. This MIB module is designed to be security-model independent, and contains objects useful for managing common aspects of any TMSM-based security model. Specific security models may define a MIB module to contain security-model-dependent information.

10.2.1. Textual Conventions

Generic and Common Textual Conventions used in this document can be found summarized at http://www.ops.ietf.org/mib-common-tcs.html

10.2.2. MIB Modules Required for IMPORTS

The following MIB module imports items from [RFC2578], [RFC2579], [RFC2580], [RFC3411], and [RFC3419]

11. Definitions

```
TMSM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    mib-2, Integer32, Unsigned32, Gauge32
      FROM SNMPv2-SMI
    TestAndIncr, StorageType, RowStatus
      FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP
      FROM SNMPv2-CONF
    SnmpSecurityModel,
    SnmpAdminString,  SnmpSecurityLevel, SnmpEngineID
       FROM SNMP-FRAMEWORK-MIB
    TransportAddress, TransportAddressType
      FROM TRANSPORT-ADDRESS-MIB
    ;

tmsmMIB MODULE-IDENTITY
    LAST-UPDATED "200604200000Z"
    ORGANIZATION "ISMS Working Group"
    CONTACT-INFO "WG-EMail:   isms@lists.ietf.org
                  Subscribe:  isms-request@lists.ietf.org

                Chairs:
                  Juergen Quittek
                  NEC Europe Ltd.
                  Network Laboratories
                  Kurfuersten-Anlage 36
                  69115 Heidelberg
                  Germany
                  +49 6221 90511-15
                   quittek@netlab.nec.de

                   Juergen Schoenwaelder
                   International University Bremen
```

Campus Ring 1
28725 Bremen
Germany
+49 421 200-3587
j.schoenwaelder@iu-bremen.de

Editor:
David Harrington
FutureWei Technologies
1700 Alma Drive, Suite 100
Plano, Texas 75075
USA
+1 603-436-8634
dharrington@huawei.com

```
                  "
        DESCRIPTION  "The Transport Mapping Security Model
                                  MIB

                     Copyright (C) The Internet Society (2006). This
                     version of this MIB module is part of RFC XXXX;
                     see the RFC itself for full legal notices.
   -- NOTE to RFC editor: replace XXXX with actual RFC number
   --                     for this document and remove this note
                     "

        REVISION     "200604200000Z"         -- 20 April 2006
        DESCRIPTION  "The initial version, published in RFC XXXX.
   -- NOTE to RFC editor: replace XXXX with actual RFC number
   --                     for this document and remove this note
                     "

       ::= { mib-2 xxxx }
   -- RFC Ed.: replace xxxx with IANA-assigned number and
   --          remove this note

   -- ------------------------------------------------------------ --
   -- subtrees in the TMSM-MIB
   -- ------------------------------------------------------------ --

   tmsmNotifications OBJECT IDENTIFIER ::= { tmsmMIB 0 }
   tmsmObjects       OBJECT IDENTIFIER ::= { tmsmMIB 1 }
   tmsmConformance   OBJECT IDENTIFIER ::= { tmsmMIB 2 }

   -- ------------------------------------------------------------
   -- Objects
   -- ------------------------------------------------------------

   -- Textual Conventions
```

```
   SessionIndex ::= TEXTUAL-CONVENTION
       DISPLAY-HINT "d"
       STATUS       current
       DESCRIPTION
           "A unique value, greater than zero, identifying a transport
            mapping security model session. The value must remain
            constant for the duration of a session. New values should
            be assigned in such a way that reuse of recently used
            values is avoided."
       SYNTAX Integer (1..2147483647)

   SessionIndexOrZero TEXTUAL-CONVENTION
       DISPLAY-HINT "d"
       STATUS       current
       DESCRIPTION
           "This extension of the TmsmSessionId permits the additional
            value zero. The meaning of the value zero is object-specific
            and must therefore be defined as part of the description of
            any object which uses this syntax. Examples of the usage of
            zero might include situations where a session was unknown
            or where none or all sessions need to be referenced."
       SYNTAX Integer (0..2147483647)

   -- Notifications for the Transport Model Security Model extension

   -- Statistics for the Transport Model Security Model extension


   tmsmStats         OBJECT IDENTIFIER ::= { tmsmObjects 1 }

   tmsmSessionOpenErrors  OBJECT-TYPE
       SYNTAX       Counter32
       MAX-ACCESS   read-only
       STATUS       current
       DESCRIPTION "The number of times an openSession() request
                   failed to open a Session.
                    "
       ::= { tmsmStats 1 }


   -- The tmsmSession Group

   tmsmSession          OBJECT IDENTIFIER ::= { tmsmObjects 2 }

   tmsmSessionSpinLock  OBJECT-TYPE
       SYNTAX       TestAndIncr
       MAX-ACCESS   read-write
       STATUS       current
```

```
        DESCRIPTION "An advisory lock used to allow several cooperating
                     TMSM security models to coordinate their
                     use of facilities to create sessions in the
                     tmsmSessionTable.
                    "
        ::= { tmsmSession 1 }

    tmsmSessionCurrent  OBJECT-TYPE
        SYNTAX       Gauge32
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The current number of open sessions.
                    "
        ::= { tmsmSession 2 }

    tmsmSessionMaxSupported  OBJECT-TYPE
        SYNTAX       Unsigned32
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The maximum number of open sessions supported.
                     The value zero indicates the maximum is dynamic.
                    "
        ::= { tmsmSession 3 }

    tmsmSessionOpenErrors  OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The number of times an openSession() request
                    failed to open a Session.
                     "
        ::= { tmsmSession 4 }

    tmsmSessionSecurityLevelNotAvailableErrors  OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The number of times an outgoing message was
                    discarded because a requested securityLevel could not
                    provided.
                     "
        ::= { tmsmSession 5 }

    tmsmSessionTable     OBJECT-TYPE
        SYNTAX       SEQUENCE OF TmsmSessionEntry
        MAX-ACCESS   not-accessible
        STATUS       current
        DESCRIPTION "The table of currently available sessions configured
```

```
                    in the SNMP engine's Local Configuration Datastore
                    (LCD).

                    Sessions are created as needed, and do not persist
                    across network management system reboots.
                    "
       ::= { tmsmSession 6 }


   tmsmSessionEntry     OBJECT-TYPE
       SYNTAX       TmsmSessionEntry
       MAX-ACCESS   not-accessible
       STATUS       current
       DESCRIPTION "A session configured in the SNMP engine's Local
                    Configuration Datastore (LCD) for Transport Mapping
                    Security Models.

                   "
       INDEX       { tmsmSessionID }
       ::= { tmsmSessionTable 1 }

   TmsmSessionEntry ::= SEQUENCE
      {
          tmsmSessionID                   SessionIndex,
          tmsmSessionTransport            TransportAddressType,
          tmsmSessionAddress              TransportAddress,
          tmsmSessionSecurityModel        SnmpSecurityModel,
          tmsmSessionSecurityName         SnmpAdminString,
          tmsmSessionSecurityLevel        SnmpSecurityLevel,
          tmsmSessionEngineID             SnmpEngineID
      }

    tmsmSessionID  OBJECT-TYPE
       SYNTAX       SessionIndex
       MAX-ACCESS   not-accessible
       STATUS       current
       DESCRIPTION "A locally-unique identifier for a session.
                   "
       ::= { tmsmSessionEntry 1 }

    tmsmSessionTransport  OBJECT-TYPE
       SYNTAX       TransportAddressType
       MAX-ACCESS   read-only
       STATUS       current
       DESCRIPTION "The transport domain associated with this session.
                   "
       ::= { tmsmSessionEntry 2 }
```

```
    tmsmSessionAddress OBJECT-TYPE
       SYNTAX       TransportAddress
       MAX-ACCESS   read-only
       STATUS       current
       DESCRIPTION "The transport address associated with this session.
                   "
       ::= { tmsmSessionEntry 3 }

   tmsmSessionSecurityModel OBJECT-TYPE
       SYNTAX       SnmpSecurityModel
       MAX-ACCESS   read-only
       STATUS       current
       DESCRIPTION "The Security Model associated with this session."
       ::= { tmsmSessionEntry 4 }

   tmsmSessionSecurityName OBJECT-TYPE
       SYNTAX       SnmpAdminString
       MAX-ACCESS   read-only
       STATUS       current
       DESCRIPTION "A human readable string representing the principal
                    in Security Model independent format.
                   "
       ::= { tmsmSessionEntry 5 }

   tmsmSessionSecurityLevel OBJECT-TYPE
       SYNTAX      SnmpSecurityLevel
       MAX-ACCESS   read-only
       STATUS       current
        DESCRIPTION "The Level of Security at which SNMP messages can be
                    sent using this session, in particular, one of:

                      noAuthNoPriv - without authentication and
                                     without privacy,
                      authNoPriv   - with authentication but
                                     without privacy,
                      authPriv     - with authentication and
                                     with privacy.
                   "
       DEFVAL      { authPriv }
       ::= { tmsmSessionEntry 6 }

   tmsmSessionEngineID  OBJECT-TYPE
       SYNTAX       SnmpEngineID
       MAX-ACCESS   read-only
       STATUS       current
       DESCRIPTION "The administratively-unique identifier for the
                    remote SNMP engine associated with this session.
                     "
```

```
        ::= { tmsmSessionEntry 7 }

    -- ------------------------------------------------------------
    -- tmsmMIB - Conformance Information
    -- ------------------------------------------------------------

    tmsmGroups OBJECT IDENTIFIER ::= { tmsmConformance 1 }

    tmsmCompliances OBJECT IDENTIFIER ::= { tmsmConformance 2 }

    -- ------------------------------------------------------------
    -- Units of conformance
    -- ------------------------------------------------------------
    tmsmGroup OBJECT-GROUP
        OBJECTS {
            tmsmSessionOpenErrors,
            tmsmSessionSecurityLevelNotAvailableErrors,
            tmsmSessionCurrent,
            tmsmSessionMaxSupported,
            tmsmSessionTransport,
            tmsmSessionAddress,
            tmsmSessionSecurityModel,
            tmsmSessionSecurityName,
            tmsmSessionSecurityLevel,
            tmsmSessionEngineID,
            tmsmSessionSpinLock
        }
        STATUS      current
        DESCRIPTION "A collection of objects for maintaining session
                     information of an SNMP engine which implements the
                     TMSM architectural extension.
                    "

        ::= { tmsmGroups 2 }

    -- ------------------------------------------------------------
    -- Compliance statements
    -- ------------------------------------------------------------

    tmsmCompliance MODULE-COMPLIANCE
        STATUS      current
        DESCRIPTION
            "The compliance statement for SNMP engines that support the
            TMSM-MIB"
        MODULE
            MANDATORY-GROUPS { tmsmGroup }
        ::= { tmsmCompliances 1 }
```

```
END
```

12. Security Considerations

This document describes an architectural approach and multiple proposed configurations that would permit SNMP to utilize transport layer security services. Each section containing a proposal should discuss the security considerations of that approach.

It is considered desirable by some industry segments that SNMP security models should utilize transport layer security that addresses perfect forward secrecy at least for encryption keys. Perfect forward secrecy guarantees that compromise of long term secret keys does not result in disclosure of past session keys.

There are a number of management objects defined in this MIB module with a MAX-ACCESS clause of read-write and/or read-create. Such objects may be considered sensitive or vulnerable in some network environments. The support for SET operations in a non-secure environment without proper protection can have a negative effect on network operations. These are the tables and objects and their sensitivity/vulnerability:

- [discuss] Should it be possible for a manager to create or modify rows in the session table? If so, then we may need the rowstatus object. If the session table is read-only then we can probably eliminate the rowstatus. If the tabel is not read-only, then we need to list the tables and objects and state why they are sensitive.

There are no management objects defined in this MIB module that have a MAX-ACCESS clause of read-write and/or read-create. So, if this MIB module is implemented correctly, then there is no risk that an intruder can alter or create any management objects of this MIB module via direct SNMP SET operations.

Some of the readable objects in this MIB module (i.e., objects with a MAX-ACCESS other than not-accessible) may be considered sensitive or vulnerable in some network environments. It is thus important to control even GET and/or NOTIFY access to these objects and possibly to even encrypt the values of these objects when sending them over the network via SNMP. These are the tables and objects and their sensitivity/vulnerability:

- o [todo] list the tables and objects and state why they are sensitive.

[discuss] how do we modify this section for an SNMP/SSH or other transport mapping security model? If the security model provides for securityName/Level/Model then some of the normal boilerplate is not true.

SNMP versions prior to SNMPv3 did not include adequate security. Even if the network itself is secure (for example by using IPSec), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB module.

It is RECOMMENDED that implementers consider the security features as provided by the SNMPv3 framework (see [RFC3410], section 8), including full support for the SNMPv3 cryptographic mechanisms (for authentication and privacy).

Further, deployment of SNMP versions prior to SNMPv3 is NOT RECOMMENDED. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security. It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.

13. IANA Considerations

The MIB module in this document uses the following IANA-assigned OBJECT IDENTIFIER values recorded in the SMI Numbers registry:

Descriptor	OBJECT IDENTIFIER value
tmsmMIB	{ mib-2 XXXX }

Editor's Note (to be removed prior to publication): the IANA is requested to assign a value for "XXXX" under the 'mib-2' subtree and to record the assignment in the SMI Numbers registry. When the assignment has been made, the RFC Editor is asked to replace "XXXX" (here and in the MIB module) with the assigned value and to remove this note.

[discuss] How do we add a new TransportType?

14. Acknowledgments

The Integrated Security for SNMP WG would like to thank the following people for their contributions to the process:

The authors of submitted security model proposals: Chris Elliot, Wes Hardaker, Dave Harrington, Keith McCloghrie, Kaushik Narayan, Dave Perkins, Joseph Salowey, and Juergen Schoenwaelder.

The members of the Protocol Evaluation Team: Uri Blumenthal, Lakshminath Dondeti, Randy Presuhn, and Eric Rescorla.

WG members who committed to and performed detailed reviews: Jeffrey Hutzelman

15. References

15.1. Normative References

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

[RFC2222] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

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

[RFC2580] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management

Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3412] Case, J., Harrington, D., Presuhn, R., and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3412, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC3417] Presuhn, R., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.

[RFC3418] Presuhn, R., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

[RFC3419] Daniele, M. and J. Schoenwaelder, "Textual Conventions for Transport Addresses", RFC 3419, December 2002.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

15.2. Informative References

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

[RFC3413] Levi, D., Meyer, P., and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.

[I-D.ietf-netconf-ssh] Wasserman, M. and T. Goddard, "Using the NETCONF Configuration Protocol over Secure Shell (SSH)", draft-ietf-netconf-ssh-06 (work in progress), March 2006.

Appendix A. Parameter Table

Following is a CSV formatted matrix useful for tracking data flows into and out of the dispatcher, message, and security subsystems. Import this into your favorite spreadsheet or other CSV compatible application. You will need to remove lines feeds from the second and

third lines, which needed to be wrapped to fit into RFC limits.

A.1. ParameterList.csv

```
,Dispatcher,,,,Messaging,,,Security,,

,sendPdu,returnResponse,processPdu,processResponse
,prepareOutgoingMessage,prepareResponseMessage,prepareDataElements
,generateRequest,processIncoming,generateResponse

transportDomain,In,,,,In,,In,,,

transportAddress,In,,,,In,,In,,,

destTransportDomain,,,,,Out,Out,,,,

destTransportAddress,,,,,Out,Out,,,,

messageProcessingModel,In,In,In,In,In,In,In,Out,In,In,In

securityModel,In,In,In,In,In,In,In,Out,In,In,In

securityName,In,In,In,In,In,In,In,Out,In,Out,In

securityLevel,In,In,In,In,In,In,In,Out,In,In,In

contextEngineID,In,In,In,In,In,In,In,Out,,,

contextName,In,In,In,In,In,In,In,Out,,,

expectResponse,In,,,,In,,,,,

PDU,In,In,In,In,In,In,In,Out,,,

pduVersion,In,In,In,In,In,In,In,Out,,,

statusInfo,Out,In,,In,,In,Out,Out,Out,Out

errorIndication,Out,Out,,,,,Out,,,

sendPduHandle,Out,,,In,In,,Out,,,

maxSizeResponsePDU,,In,In,,,In,Out,,Out,

stateReference,,In,In,,,In,Out,,,

wholeMessage,,,,,Out,Out,,Out,In,Out
```

messageLength,,,,,Out,Out,,Out,In,Out

maxMessageSize,,,,,,,,In,In,In

globalData,,,,,,,,In,,In

securityEngineID,,,,,,,,In,Out,In

scopedPDU,,,,,,,,In,Out,In

securityParameters,,,,,,,,Out,,Out

securityStateReference,,,,,,,,,Out,In

pduType,,,,,,,Out,,,

tmStateReference,,,,,,Out,In,,In,

Appendix B. Why tmSecurityReference?

This appendix considers why a cache-based approach was selected for passing parameters. This section may be removed from subsequent revisions fo the document.

There are four approaches that could be used for passing information between the TMSP and an MPSP.

1. one could define an ASI to supplement the existing ASIs, or
2. the TMSM could add a header to encapsulate the SNMP message,
3. the TMSM could utilize fields already defined in the existing SNMPv3 message, or
4. the TMSM could pass the information in an implementation-specific cache or via a MIB module.

B.1. Define an Abstract Service Interface

Abstract Service Interfaces (ASIs) [RFC3411] are defined by a set of primitives that specify the services provided and the abstract data elements that are to be passed when the services are invoked. Defining additional ASIs to pass the security and transport information from the transport mapping to a messaging security model has the advantage of being consistent with existing RFC3411/3412 practice, and helps to ensure that any TMSM proposals pass the necessary data, and do not cause side effects by creating model-specific dependencies between itself and other models or other subsystems other than those that are clearly defined by an ASI.

B.2. Using an Encapsulating Header

A header could encapsulate the SNMP message to pass necessary information from the TMSP to the dispatcher and then to a messaging security model. The message header would be included in the wholeMessage ASI parameter, and would be removed by a corresponding messaging model. This would imply the (one and only) messaging dispatcher would need to be modified to determine which SNMP message version was involved, and a new message processing model would need to be developed that knew how to extract the header from the message and pass it to the MPSP.

B.3. Modifying Existing Fields in an SNMP Message

[RFC3412] describes the SNMPv3 message, which contains fields to pass security related parameters. The TMSM could use these fields in an SNMPv3 message, or comparable fields in other message formats to pass information between transport mapping security models in different SNMP engines, and to pass information between a transport mapping security model and a corresponding messaging security model.

If the fields in an incoming SNMPv3 message are changed by the TMSP before passing it to the MPSP, then the TMSP will need to decode the ASN.1 message, modify the fields, and re-encode the message in ASN.1 before passing the message on to the message dispatcher or to the transport layer. This would require an intimate knowledge of the message format and message versions so the TMSP knew which fields could be modified. This would seriously violate the modularity of the architecture.

B.4. Using a Cache

This document describes a cache, into which the TMSP puts information about the security applied to an incoming message, and an MPSP extracts that information from the cache. Given that there may be multiple TM-security caches, a tmStateReference is passed as an extra parameter in the ASIs between the transport mapping and the messaging security model.so the MPSP knows which cache of information to consult.

This approach does create dependencies between a model-specific TMSP and a corresponding specific MPSP. This approach of passing a model-independent reference is consistent with the securityStateReference cache already being passed around in the RFC3411 ASIs.

Appendix C. Open Issues

Appendix D. Change Log

NOTE to RFC editor: Please remove this change log before publishing this document as an RFC.

Changes from revision -01- to -02-

- wrote text for session establishment requirements section.
- wrote text for session maintenance requirements section.
- removed section on relation to SNMPv2-MIB
- updated MIB module to pass smilint
- Added Structure of the MIB module, and other expected MIB-related sections.
- updated author address
- corrected spelling
- removed msgFlags appendix
- Removed section on implementation considerations.
- started modifying the security boilerplate to address TMSM and MIB security issues
- reorganized slightly to better separate requirements from proposed solution. This probably needs additional work.
- removed section with sample protocols and sample tmStateReference.
- Added section for acronyms
- moved section comparing parameter passing techniques to appendix.
- Removed section on notification requirements.

Changes from revision -00-

- changed SSH references from I-Ds to RFCs
- removed parameters from tmStateReference for DTLS that revealed lower layer info.
- Added TMSM-MIB module
- Added Internet-Standard Management Framework boilerplate
- Added Structure of the MIB Module
- Added MIB security considerations boilerplate (to be completed)
- Added IANA Considerations
- Added ASI Parameter table
- Added discussion of Sessions
- Added Open issues and Change Log
- Rearranged sections

Authors' Addresses

David Harrington
Futurewei Technologies
1700 Alma Dr. Suite 100
Plano, TX 75075
USA

Phone: +1 603 436 8634
EMail: dharrington@huawei.com

Juergen Schoenwaelder
International University Bremen
Campus Ring 1
28725 Bremen
Germany

Phone: +49 421 200-3587
EMail: j.schoenwaelder@iu-bremen.de

Full Copyright Statement

Intellectual Property

Acknowledgement

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