[Note that this file is a concatenation of more than one RFC.]

Network Working Group
Request for Comments: 5343
Updates: 3411
Category: Standards Track

J. Schoenwaelder
Jacobs University Bremen

September 2008

# Simple Network Management Protocol (SNMP) Context EngineID Discovery

## Status of This Memo



## Abstract

The Simple Network Management Protocol (SNMP) version three (SNMPv3) requires that an application know the identifier (snmpEngineID) of the remote SNMP protocol engine in order to retrieve or manipulate objects maintained on the remote SNMP entity.

This document introduces a well-known localEngineID and a discovery mechanism that can be used to learn the snmpEngineID of a remote SNMP protocol engine. The proposed mechanism is independent of the features provided by SNMP security models and may also be used by other protocol interfaces providing access to managed objects.

This document updates RFC 3411.

## Table of Contents

## 1. Introduction

To retrieve or manipulate management information using the third version of the Simple Network Management Protocol (SNMPv3) [RFC3410], it is necessary to know the identifier of the remote SNMP protocol engine, the so-called snmpEngineID [RFC3411]. While an appropriate snmpEngineID can in principle be configured on each management application for each SNMP agent, it is often desirable to discover the snmpEngineID automatically.

This document introduces a discovery mechanism that can be used to learn the snmpEngineID of a remote SNMP protocol engine. The proposed mechanism is independent of the features provided by SNMP security models. The mechanism has been designed to coexist with discovery mechanisms that may exist in SNMP security models, such as the authoritative engine identifier discovery of the User-based Security Model (USM) of SNMP [RFC3414].

This document updates RFC 3411 [RFC3411] by clarifying the IANA rules for the maintenance of the SnmpEngineID format registry.

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].

## 2. Background

Within an administrative domain, an SNMP engine is uniquely identified by an snmpEngineID value [RFC3411]. An SNMP entity, which consists of an SNMP engine and several SNMP applications, may provide access to multiple contexts.

An SNMP context is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context and an SNMP entity potentially has access to many contexts [RFC3411]. A context is identified by the snmpEngineID value of the entity hosting the management information (also called a contextEngineID) and a context name that identifies the specific context (also called a contextName).

To identify an individual item of management information within an administrative domain, a four tuple is used consisting of

1. a contextEngineID,

2. a contextName,

3. an object type, and

4. its instance identification.

The last two elements are encoded in an object identifier (OID) value. The contextName is a character string (following the SnmpAdminString textual convention of the SNMP-FRAMEWORK-MIB [RFC3411]) while the contextEngineID is an octet string constructed according to the rules defined as part of the SnmpEngineID textual convention of the SNMP-FRAMEWORK-MIB [RFC3411].

The SNMP protocol operations and the protocol data units (PDUs) operate on OIDs and thus deal with object types and instances [RFC3416]. The SNMP architecture [RFC3411] introduces the concept of a scopedPDU as a data structure containing a contextEngineID, a contextName, and a PDU. The SNMP version 3 (SNMPv3) message format uses ScopedPDUs to exchange management information [RFC3412].

Within the SNMP framework, contextEngineIDs serve as end-to-end identifiers. This becomes important in situations where SNMP proxies are deployed to translate between protocol versions or to cross middleboxes such as network address translators. In addition, snmpEngineIDs separate the identification of an SNMP engine from the transport addresses used to communicate with an SNMP engine. This property can be used to correlate management information easily, even in situations where multiple different transports were used to retrieve the information or where transport addresses can change dynamically.

To retrieve data from an SNMPv3 agent, it is necessary to know the appropriate contextEngineID. The User-based Security Model (USM) of SNMPv3 provides a mechanism to discover the snmpEngineID of the remote SNMP engine, since this is needed for security processing reasons. The discovered snmpEngineID can subsequently be used as a contextEngineID in a ScopedPDU to access management information local to the remote SNMP engine. Other security models, such as the Transport Security Model (TSM) [TSM], lack such a procedure and may use the discovery mechanism defined in this memo.

## 3. Procedure

The proposed discovery mechanism consists of two parts, namely (i) the definition of a special well-known snmpEngineID value, called the localEngineID, which always refers to a local default context, and (ii) the definition of a procedure to acquire the snmpEngineID scalar of the SNMP-FRAMEWORK-MIB [RFC3411] using the special well-known local localEngineID value.

### 3.1. Local EngineID

An SNMP command responder implementing this specification MUST register their pduTypes using the localEngineID snmpEngineID value (defined below) by invoking the registerContextEngineID() Abstract Service Interface (ASI) defined in RFC 3412 [RFC3412]. This registration is done in addition to the normal registration under the SNMP engine's snmpEngineID. This is consistent with the SNMPv3 specifications since they explicitly allow registration of multiple engineIDs and multiple pduTypes [RFC3412].

The SnmpEngineID textual convention [RFC3411] defines that an snmpEngineID value MUST be between 5 and 32 octets long. This specification proposes to use the variable length format 3) of the SnmpEngineID textual convention and to allocate the reserved, unused format value 6, using the enterprise ID 0 for the localEngineID. An ASN.1 definition for localEngineID would look like this:

```
localEngineID OCTET STRING ::= '8000000006'H
```

The localEngineID value always provides access to the default context of an SNMP engine. Note that the localEngineID value is intended to be used as a special value for the contextEngineID field in the ScopedPDU. It MUST NOT be used as a value to identify an SNMP engine; that is, this value MUST NOT be used in the snmpEngineID.0 scalar [RFC3418] or in the msgAuthoritativeEngineID field in the securityParameters of the User-based Security Model (USM) [RFC3414].

### 3.2. EngineID Discovery

Discovery of the snmpEngineID is done by sending a Read Class protocol operation (see Section 2.8 of [RFC3411]) to retrieve the snmpEngineID scalar using the localEngineID defined above as a contextEngineID value. Implementations SHOULD only perform this discovery step when it is needed. In particular, if security models are used that already discover the remote snmpEngineID (such as USM), then no further discovery is necessary. The same is true in situations where the application already knows a suitable snmpEngineID value.

The procedure to discover the snmpEngineID of a remote SNMP engine can be described as follows:

1. Check whether a suitable contextEngineID value is already known. If yes, use the provided contextEngineID value and stop the discovery procedure.

2. Check whether the selected security model supports discovery of the remote snmpEngineID (e.g., USM with its discovery mechanism). If yes, let the security model perform the discovery. If the remote snmpEngineID value has been successfully determined, assign it to the contextEngineID and stop the discovery procedure.

3. Send a Read Class operation to the remote SNMP engine using the localEngineID value as the contextEngineID in order to retrieve the scalar snmpEngineID.0 of the SNMP-FRAMEWORK-MIB [RFC3411]. If successful, set the contextEngineID to the retrieved value and stop the discovery procedure.

4. Return an error indication that a suitable contextEngineID could not be discovered.

The procedure outlined above is an example and can be modified to retrieve more variables in step 3, such as the sysObjectID.0 scalar or the snmpSetSerialNo.0 scalar of the SNMPv2-MIB [RFC3418].

## 4. IANA Considerations

RFC 3411 requested that IANA create a registry for SnmpEngineID formats. However, RFC 3411 did not ask IANA to record the initial assignments made by RFC 3411 nor did RFC 3411 spell out the precise allocation rules. To address this issue, the following rules are hereby established.

IANA maintains a registry for SnmpEngineID formats. The first four octets of an SnmpEngineID carry an enterprise number, while the fifth octet in a variable length SnmpEngineID value, called the format octet, indicates how the following octets are formed. The following format values were allocated in [RFC3411]:

| Format | Description | References |
|---|---|---|
| 0 | reserved, unused | [RFC3411] |
| 1 | IPv4 address | [RFC3411] |
| 2 | IPv6 address | [RFC3411] |
| 3 | MAC address | [RFC3411] |
| 4 | administratively assigned text | [RFC3411] |
| 5 | administratively assigned octets | [RFC3411] |
| 6-127 | reserved, unused | [RFC3411] |
| 128-255 | enterprise specific | [RFC3411] |

IANA can assign new format values out of the originally assigned and reserved number space 1-127. For new assignments in this number

space, a specification is required as per [RFC5226]. The number space 128-255 is enterprise specific and is not controlled by IANA.

Per this document, IANA has made the following assignment:

| Format | Description | References |
|---|---|---|
| 6 | local engine | [RFC5343] |

## 5. Security Considerations

SNMP version 3 (SNMPv3) provides cryptographic security to protect devices from unauthorized access. This specification recommends use of the security services provided by SNMPv3. In particular, it is RECOMMENDED to protect the discovery exchange.

An snmpEngineID can contain information such as a device's MAC address, IPv4 address, IPv6 address, or administratively assigned text. An attacker located behind a router / firewall / network address translator may not be able to obtain this information directly, and he therefore might discover snmpEngineID values in order to obtain this kind of device information.

In many environments, making snmpEngineID values accessible via a security level of noAuthNoPriv will benefit legitimate tools that try to algorithmically determine some basic information about a device. For this reason, the default View-based Access Control Model (VACM) configuration in Appendix A of RFC 3415 [RFC3415] gives noAuthNoPriv read access to the snmpEngineID. Furthermore, the USM discovery mechanism defined in RFC 3414 [RFC3414] uses unprotected messages and reveals snmpEngineID values.

In highly secure environments, snmpEngineID values can be protected by using the discovery mechanism described in this document together with a security model that does not exchange cleartext SNMP messages, such as the Transport Security Model (TSM) [TSM].

The isAccessAllowed() abstract service primitive of the SNMP access control subsystem does not take the contextEngineID into account when checking access rights [RFC3411]. As a consequence, it is not possible to define a special view for context engineID discovery. A request with a localEngineID is thus treated like a request with the correct snmpEngineID by the access control subsystem. This is inline with the SNMPv3 design where the authenticated identity is the securityName (together with the securityModel and securityLevel information), and transport addresses or knowledge of contextEngineID values do not impact the access-control decision.

## 6. Acknowledgments

Dave Perkins suggested the introduction of a "local" contextEngineID during the interim meeting of the ISMS (Integrated Security Model for SNMP) working group in Boston, 2006. Joe Fernandez, David Harrington, Dan Romascanu, and Bert Wijnen provided helpful review and feedback, which helped to improve this document.

## 7. References

### 7.1. Normative References

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

[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.

[RFC3416] Presuhn, R., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

### 7.2. 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.

[RFC3415] Wijnen, B., Presuhn, R., and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3415, December 2002.

[TSM] Harrington, D., "Transport Security Model for SNMP", Work in Progress, July 2008.

## Author's Address

Juergen Schoenwaelder
Jacobs University Bremen
Campus Ring 1
28725 Bremen
Germany

Phone: +49 421 200-3587
EMail: j.schoenwaelder@jacobs-university.de

## Full Copyright Statement



## Intellectual Property

Network Working Group
Request for Comments: 5590
Updates: 3411, 3412, 3414, 3417
Category: Standards Track

D. Harrington
Huawei Technologies (USA)
J. Schoenwaelder
Jacobs University Bremen

June 2009

# Transport Subsystem for the Simple Network Management Protocol (SNMP)

## Status of This Memo



## Copyright Notice



## Abstract

This document defines a Transport Subsystem, extending the Simple Network Management Protocol (SNMP) architecture defined in RFC 3411. This document defines a subsystem to contain Transport Models that is comparable to other subsystems in the RFC 3411 architecture. As work is being done to expand the transports to include secure transports, such as the Secure Shell (SSH) Protocol and Transport Layer Security

(TLS), using a subsystem will enable consistent design and modularity of such Transport Models. This document identifies and describes some key aspects that need to be considered for any Transport Model for SNMP.

Table of Contents

## 1. Introduction

This document defines a Transport Subsystem, extending the Simple Network Management Protocol (SNMP) architecture defined in [RFC3411]. This document identifies and describes some key aspects that need to be considered for any Transport 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].

### 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].

Lowercase versions of the keywords should be read as in normal English. They will usually, but not always, be used in a context that relates to compatibility with the RFC 3411 architecture or the subsystem defined here but that might have no impact on on-the-wire compatibility. These terms are used as guidance for designers of proposed IETF models to make the designs compatible with RFC 3411 subsystems and Abstract Service Interfaces (ASIs). Implementers are free to implement differently. Some usages of these lowercase terms are simply normal English usage.

For consistency with SNMP-related specifications, this document favors terminology as defined in STD 62, rather than favoring terminology that is consistent with non-SNMP specifications that use different variations of the same terminology. This is consistent with the IESG decision to not require the SNMPv3 terminology be modified to match the usage of other non-SNMP specifications when SNMPv3 was advanced to Full Standard.

This document discusses an extension to the modular RFC 3411 architecture; this is not a protocol document. An architectural "MUST" is a really sharp constraint; to allow for the evolution of technology and to not unnecessarily constrain future models, often a

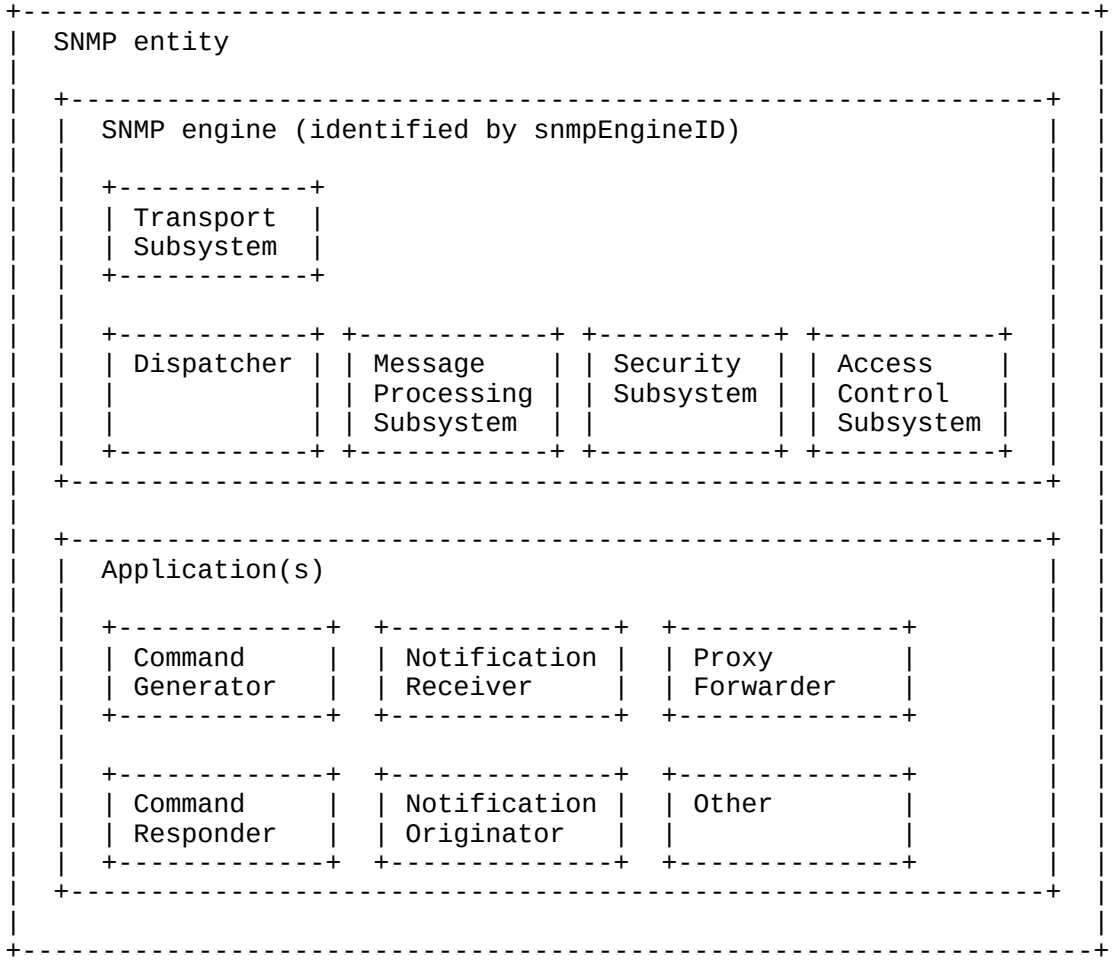
"SHOULD" or a "should" is more appropriate than a "MUST" in an architecture. Future models MAY express tighter requirements for their own model-specific processing.

## 1.3. Where This Extension Fits

It is expected that readers of this document will have read RFCs 3410 and 3411, and have a general understanding of the functionality defined in RFCs 3412-3418.

The "Transport Subsystem" is an additional component for the SNMP Engine depicted in RFC 3411, Section 3.1.

The following diagram depicts its place in the RFC 3411 architecture.



The transport mappings defined in RFC 3417 do not provide lower-layer security functionality, and thus do not provide transport-specific security parameters. This document updates RFC 3411 and RFC 3417 by defining an architectural extension and modifying the ASIs that transport mappings (hereafter called "Transport Models") can use to pass transport-specific security parameters to other subsystems, including transport-specific security parameters that are translated into the transport-independent securityName and securityLevel parameters.

The Transport Security Model [RFC5591] and the Secure Shell Transport Model [RFC5592] utilize the Transport Subsystem. The Transport Security Model is an alternative to the existing SNMPv1 Security Model [RFC3584], the SNMPv2c Security Model [RFC3584], and the User-based Security Model [RFC3414]. The Secure Shell Transport Model is an alternative to existing transport mappings as described in [RFC3417].

## 2. Motivation

Just as there are multiple ways to secure one’s home or business, in a continuum of alternatives, there are multiple ways to secure a network management protocol. Let’s consider three general approaches.

In the first approach, an individual could sit on his front porch waiting for intruders. In the second approach, he could hire an employee, schedule the employee, position the employee to guard what he wants protected, hire a second guard to cover if the first gets sick, and so on. In the third approach, he could hire a security company, tell them what he wants protected, and leave the details to them. Considerations of hiring and training employees, positioning and scheduling the guards, arranging for cover, etc., are the responsibility of the security company. The individual therefore achieves the desired security, with significantly less effort on his part except for identifying requirements and verifying the quality of 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 (e.g., SHA), 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 uses a separate principal and key management infrastructure. Operators have reported that deploying another principal and key management infrastructure in order to use

SNMPv3 is a deterrent to deploying SNMPv3. It is possible to use external mechanisms to handle the distribution of keys for use by USM. The more important issue is that operators wanted to leverage existing user management infrastructures that were not specific to SNMP.

A USM-compliant architecture might combine the authentication mechanism with an external mechanism, such as RADIUS [RFC2865], to provide the authentication service. Similarly, it might be possible to utilize an external protocol to encrypt a message, to check timeliness, to check data integrity, etc. However, this corresponds to the second approach -- requiring the coordination of a number of differently subcontracted services. Building solid security between the various services is difficult, and there is a significant potential for gaps in security.

An alternative approach might be to 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. This corresponds to the third approach described above. 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 are TLS [RFC5246], Simple Authentication and Security Layer (SASL) [RFC4422], 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 other network management protocols, such as syslog [RFC5424] and NETCONF [RFC4741].

This document defines a Transport Subsystem extension to the RFC 3411 architecture that is based on the third approach. This extension specifies how other lower-layer protocols with common security infrastructures can be used underneath the SNMP protocol and the desired goal of unified administrative security can be met.

This extension allows security to be provided by an external protocol connected to the SNMP engine through an SNMP Transport Model [RFC3417]. Such a Transport Model would then enable the use of existing security mechanisms, such as TLS [RFC5246] or SSH [RFC4251], within the RFC 3411 architecture.

There are a number of Internet security protocols and mechanisms that are in widespread use. Many of them try to provide a generic infrastructure to be used by many different application-layer protocols. The motivation behind the Transport Subsystem 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 provide interoperability with existing implementations. These challenges are described in detail in this document. For some key issues, design choices are described that might be made to provide a workable solution that meets operational requirements and fits into the SNMP architecture defined in [RFC3411].

## 3. Requirements of a Transport Model

### 3.1. Message Security Requirements

Transport security protocols SHOULD provide protection against the following message-oriented threats:

1. modification of information

2. masquerade

3. message stream modification

4. disclosure

These threats are described in Section 1.4 of [RFC3411]. The security requirements outlined there do not require protection against denial of service or traffic analysis; however, transport security protocols should not make those threats significantly worse.

#### 3.1.1. Security Protocol Requirements

There are a number of standard protocols that could be proposed as possible solutions within the Transport Subsystem. Some factors should be considered when selecting a protocol.

Using a protocol in a manner for which it was not designed has numerous problems. The advertised security characteristics of a protocol might depend on it being used as designed; when used in other ways, it might not deliver the expected security characteristics. It is recommended that any proposed model include a description of the applicability of the Transport Model.

A Transport Model SHOULD NOT require modifications to the underlying protocol. Modifying the protocol might change its security characteristics in ways that could impact other existing usages. If a change is necessary, the change SHOULD be an extension that has no impact on the existing usages. Any Transport Model specification should include a description of potential impact on other usages of the protocol.

Since multiple Transport Models can exist simultaneously within the Transport Subsystem, Transport Models MUST be able to coexist with each other.

## 3.2. SNMP Requirements

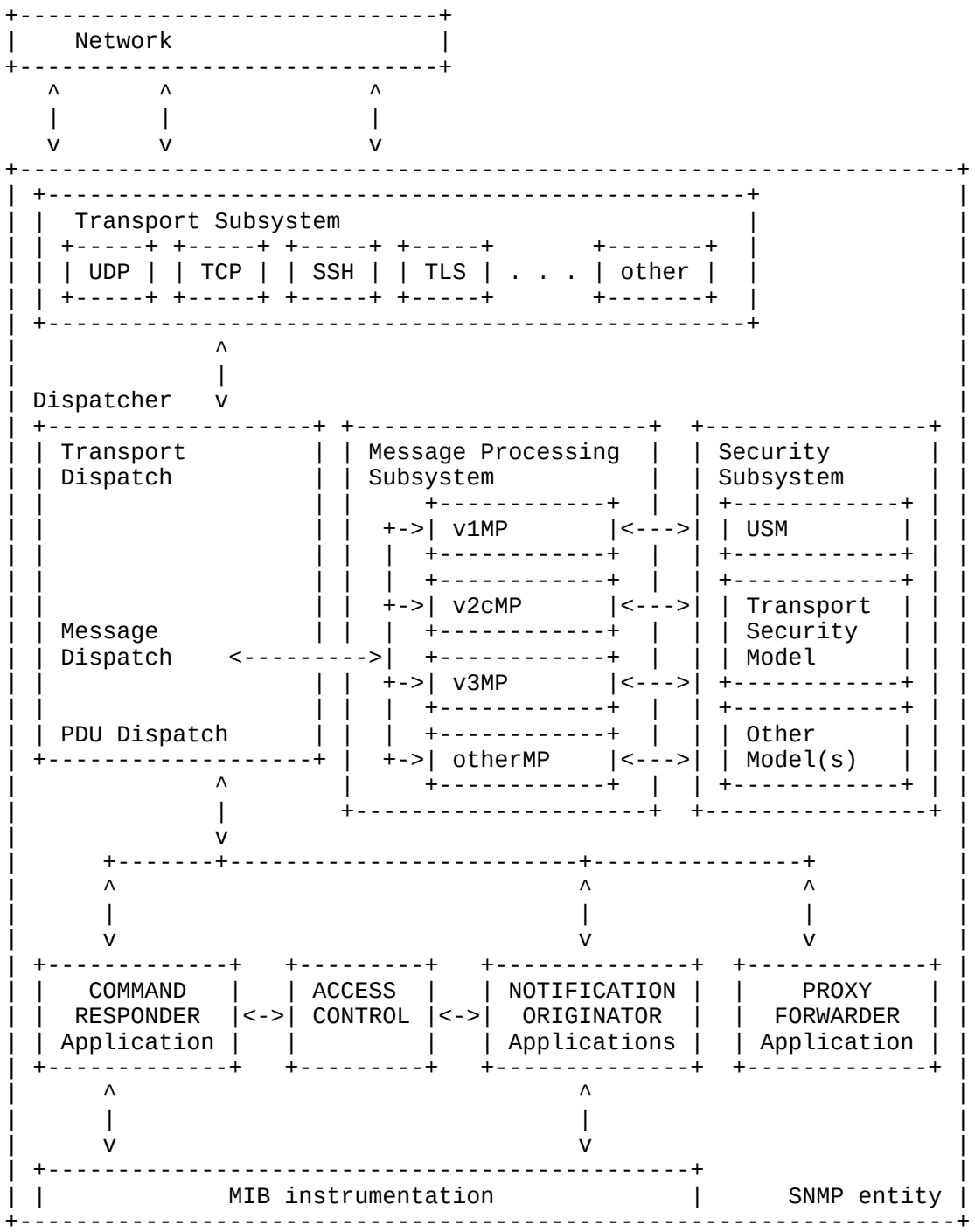
### 3.2.1. Architectural Modularity Requirements

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

The RFC 3411 architecture includes a Message Processing Subsystem for permitting different message versions to be handled by a single engine, a Security Subsystem for enabling different methods of providing security services, Applications to support different types of Application processors, and an Access Control Subsystem for allowing multiple approaches to access control. The RFC 3411 architecture does not include a subsystem for Transport Models, despite the fact there are multiple transport mappings already defined for SNMP [RFC3417]. This document describes a Transport Subsystem that is compatible with the RFC 3411 architecture. As work is being done to use secure transports such as SSH and TLS, using a subsystem will enable consistent design and modularity of such Transport Models.

The design of this Transport Subsystem accepts the goals of the RFC 3411 architecture that are defined in Section 1.5 of [RFC3411]. This Transport Subsystem uses a modular design that permits Transport Models (which might or might not be security-aware) to be "plugged into" the RFC 3411 architecture. Such Transport Models would be independent of other modular SNMP components as much as possible. This design also permits Transport Models to be advanced through the standards process independently of other Transport Models.

The following diagram depicts the SNMPv3 architecture, including the new Transport Subsystem defined in this document and a new Transport Security Model defined in [RFC5591].


Network
Transport Subsystem
UDP
TCP
SSH
TLS
. . .
other
Dispatcher
Transport Dispatch
Message Dispatch
PDU Dispatch
Message Processing Subsystem
v1MP
v2cMP
v3MP
otherMP
Security Subsystem
USM
Transport Security Model
Other Model(s)
COMMAND RESPONDER Application
ACCESS CONTROL
NOTIFICATION ORIGINATOR Applications
PROXY FORWARDER Application
MIB instrumentation
SNMP entity

#### 3.2.1.1. Changes to the RFC 3411 Architecture

The RFC 3411 architecture and the Security Subsystem assume that a Security Model is called by a Message Processing Model and will perform multiple security functions within the Security Subsystem. A Transport Model that supports a secure transport protocol might perform similar security functions within the Transport Subsystem, including the translation of transport-security parameters to/from Security-Model-independent parameters.

To accommodate this, an implementation-specific cache of transport-specific information will be described (not shown), and the data flows on this path will be extended to pass Security-Model-independent values. This document amends some of the ASIs defined in RFC 3411; these changes are covered in Section 6 of this document.

New Security Models might be defined that understand how to work with these modified ASIs and the transport-information cache. One such Security Model, the Transport Security Model, is defined in [RFC5591].

#### 3.2.1.2. Changes to RFC 3411 Processing

The introduction of secure transports affects the responsibilities and order of processing within the RFC 3411 architecture. While the steps are the same, they might occur in a different order, and might be done by different subsystems. With the existing RFC 3411 architecture, security processing starts when the Message Processing Model decodes portions of the encoded message to extract parameters that identify which Security Model MUST handle the security-related tasks.

A secure transport performs those security functions on the message, before the message is decoded. Some of these functions might then be repeated by the selected Security Model.

#### 3.2.1.3. Passing Information between SNMP Engines

A secure Transport Model will establish an authenticated and possibly encrypted tunnel between the Transport Models of two SNMP engines. After a transport-layer tunnel is established, then SNMP messages can be sent through the tunnel from one SNMP engine to the other. While the Community Security Models [RFC3584] and the User-based Security Model establish a security association for each SNMP message, newer Transport Models MAY support sending multiple SNMP messages through the same tunnel to amortize the costs of establishing a security association.

### 3.2.2. Access Control Requirements

RFC 3411 made some design decisions related to the support of an Access Control Subsystem. These include establishing and passing in a model-independent manner the securityModel, securityName, and securityLevel parameters, and separating message authentication from data-access authorization.

#### 3.2.2.1. securityName and securityLevel Mapping

SNMP data-access controls are expected to work on the basis of who can perform what operations on which subsets of data, and based on the security services that will be provided to secure the data in transit. The securityModel and securityLevel parameters establish the protections for transit -- whether authentication and privacy services will be or have been applied to the message. The securityName is a model-independent identifier of the security "principal".

A Security Model plays a role in security that goes beyond protecting the message -- it provides a mapping between the Security-Model-specific principal for an incoming message to a Security-Model independent securityName that can be used for subsequent processing, such as for access control. The securityName is mapped from a mechanism-specific identity, and this mapping must be done for incoming messages by the Security Model before it passes securityName to the Message Processing Model via the processIncoming ASI.

A Security Model is also responsible to specify, via the securityLevel parameter, whether incoming messages have been authenticated and encrypted, and to ensure that outgoing messages are authenticated and encrypted based on the value of securityLevel.

A Transport Model MAY provide suggested values for securityName and securityLevel. A Security Model might have multiple sources for determining the principal and desired security services, and a particular Security Model might or might not utilize the values proposed by a Transport Model when deciding the value of securityName and securityLevel.

Documents defining a new transport domain MUST define a prefix that MAY be prepended to all securityNames passed by the Security Model. The prefix MUST include one to four US-ASCII alpha-numeric characters, not including a ":" (US-ASCII 0x3a) character. If a prefix is used, a securityName is constructed by concatenating the prefix and a ":" (US-ASCII 0x3a) character, followed by a non-empty identity in an snmpAdminString-compatible format. The prefix can be used by SNMP Applications to distinguish "alice" authenticated by SSH

from "alice" authenticated by TLS. Transport domains and their corresponding prefixes are coordinated via the IANA registry "SNMP Transport Domains".

### 3.2.3. Security Parameter Passing Requirements

A Message Processing Model might unpack SNMP-specific security parameters from an incoming message before calling a specific Security Model to handle the security-related processing of the message. When using a secure Transport Model, some security parameters might be extracted from the transport layer by the Transport Model before the message is passed to the Message Processing Subsystem.

This document describes a cache mechanism (see Section 5) into which the Transport Model puts information about the transport and security parameters applied to a transport connection or an incoming message; a Security Model might extract that information from the cache. A tmStateReference is passed as an extra parameter in the ASIs between the Transport Subsystem and the Message Processing and Security Subsystems in order to identify the relevant cache. This approach of passing a model-independent reference is consistent with the securityStateReference cache already being passed around in the RFC 3411 ASIs.

### 3.2.4. Separation of Authentication and Authorization

The RFC 3411 architecture defines a separation of authentication and the authorization to access and/or modify MIB data. A set of model-independent parameters (securityModel, securityName, and securityLevel) are passed between the Security Subsystem, the Applications, and the Access Control Subsystem.

This separation was a deliberate decision of the SNMPv3 WG, in order to allow support for authentication protocols that do not provide data-access authorization capabilities, and in order to support data-access authorization schemes, such as the View-based access Control Model (VACM), that do not perform their own authentication.

A Message Processing Model determines which Security Model is used, either based on the message version (e.g., SNMPv1 and SNMPv2c) or possibly by a value specified in the message (e.g., msgSecurityModel field in SNMPv3).

The Security Model makes the decision which securityName and securityLevel values are passed as model-independent parameters to an Application, which then passes them via the isAccessAllowed ASI to the Access Control Subsystem.

An Access Control Model performs the mapping from the model-independent security parameters to a policy within the Access Control Model that is Access-Control-Model-dependent.

A Transport Model does not know which Security Model will be used for an incoming message, and so cannot know how the securityName and securityLevel parameters will be determined. It can propose an authenticated identity (via the tmSecurityName field), but there is no guarantee that this value will be used by the Security Model. For example, non-transport-aware Security Models will typically determine the securityName (and securityLevel) based on the contents of the SNMP message itself. Such Security Models will simply not know that the tmStateReference cache exists.

Further, even if the Transport Model can influence the choice of securityName, it cannot directly determine the authorization allowed to this identity. If two different Transport Models each authenticate a transport principal that are then both mapped to the same securityName, then these two identities will typically be afforded exactly the same authorization by the Access Control Model.

The only way for the Access Control Model to differentiate between identities based on the underlying Transport Model would be for such transport-authenticated identities to be mapped to distinct securityNames. How and if this is done is Security-Model-dependent.

### 3.3. Session Requirements

Some secure transports have a notion of sessions, while other secure transports provide channels or other session-like mechanisms. Throughout this document, the term "session" is used in a broad sense to cover transport sessions, transport channels, and other transport-layer, session-like mechanisms. Transport-layer sessions that can secure multiple SNMP messages within the lifetime of the session are considered desirable because the cost of authentication can be amortized over potentially many transactions. How a transport session is actually established, opened, closed, or maintained is specific to a particular Transport Model.

To reduce redundancy, this document describes aspects that are expected to be common to all Transport Model sessions.

#### 3.3.1. No SNMP Sessions

The architecture defined in [RFC3411] and the Transport Subsystem defined in this document do not support SNMP sessions or include a session selector in the Abstract Service Interfaces.

The Transport Subsystem might support transport sessions. However, the Transport Subsystem does not have access to the pduType (i.e., the SNMP operation type), and so cannot select a given transport session for particular types of traffic.

Certain parameters of the Abstract Service Interfaces might be used to guide the selection of an appropriate transport session to use for a given request by an Application.

The transportDomain and transportAddress identify the transport connection to a remote network node. Elements of the transport address (such as the port number) might be used by an Application to send a particular PDU type to a particular transport address. For example, the SNMP-TARGET-MIB and SNMP-NOTIFICATION-MIB [RFC3413] are used to configure notification originators with the destination port to which SNMPv2-Trap PDUs or Inform PDUs are to be sent, but the Transport Subsystem never looks inside the PDU.

The securityName identifies which security principal to communicate with at that address (e.g., different Network Management System (NMS) applications), and the securityLevel might permit selection of different sets of security properties for different purposes (e.g., encrypted SET vs. non-encrypted GET operations).

However, because the handling of transport sessions is specific to each Transport Model, some Transport Models MAY restrict selecting a particular transport session. A user application might use a unique combination of transportDomain, transportAddress, securityModel, securityName, and securityLevel to try to force the selection of a given transport session. This usage is NOT RECOMMENDED because it is not guaranteed to be interoperable across implementations and across models.

Implementations SHOULD be able to maintain some reasonable number of concurrent transport sessions, and MAY provide non-standard internal mechanisms to select transport sessions.

### 3.3.2. Session Establishment Requirements

SNMP Applications provide the transportDomain, transportAddress, securityName, and securityLevel to be used to create a new session.

If the Transport Model cannot provide at least the requested level of security, the Transport Model should discard the message and should notify the Dispatcher that establishing a session and sending the message failed. Similarly, if the session cannot be established, then the message should be discarded and the Dispatcher notified.

Transport session establishment might require provisioning authentication credentials at an engine, either statically or dynamically. How this is done is dependent on the Transport Model and the implementation.

### 3.3.3. Session Maintenance Requirements

A Transport Model can tear down sessions as needed. It might 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. How an implementation determines that an operation has completed is implementation-dependent. While it is possible to tear down each transport session after processing for each message has completed, this is not recommended for performance reasons.

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

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

### 3.3.4. Message Security versus Session Security

A Transport Model session is associated with state information that is maintained for its lifetime. This state information allows for the application of various security services to multiple messages. Cryptographic keys associated with the transport session SHOULD be used to provide authentication, integrity checking, and encryption services, as needed, for data that is communicated during the session. The cryptographic protocols used to establish keys for a Transport Model session SHOULD ensure that fresh new session keys are generated for each session. This would ensure that a cross-session replay attack would be unsuccessful; that is, an attacker could not take a message observed on one session and successfully replay it on another session.

A good security protocol would also protect against replay attacks within a session; that is, an attacker could not take a message observed on a session and successfully replay it later in the same session. One approach would be to use sequence information within the protocol, allowing the participants to detect if messages were replayed or reordered within a session.

If a secure transport session is closed between the time a request message is received and the corresponding response message is sent, then the response message SHOULD be discarded, even if a new session has been established. The SNMPv3 WG decided that this should be a "SHOULD" architecturally, and it is a Security-Model-specific decision whether to REQUIRE this. The architecture does not mandate this requirement in order to allow for future Security Models where this might make sense; however, not requiring this could lead to added complexity and security vulnerabilities, so most Security Models SHOULD require this.

SNMPv3 was designed to support multiple levels of security, selectable on a per-message basis by an SNMP Application, because, for example, there is not much value in using encryption for a command generator to poll for potentially non-sensitive performance data on thousands of interfaces every ten minutes; such encryption might add significant overhead to processing of the messages.

Some Transport Models might 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. A Transport Model MAY upgrade the security level requested by a transport-aware Security Model, i.e., noAuthNoPriv and authNoPriv might be sent over an authenticated and encrypted session. A Transport Model MUST NOT downgrade the security level requested by a transport-aware Security Model, and SHOULD discard any message where this would occur. This is a SHOULD rather than a MUST only to permit the potential development of models that can perform error-handling in a manner that is less severe than discarding the message. However, any model that does not discard the message in this circumstance should have a clear justification for why not discarding will not create a security vulnerability.

## 4. Scenario Diagrams and the Transport Subsystem

Sections 4.6.1 and 4.6.2 of RFC 3411 provide scenario diagrams to illustrate how an outgoing message is created and how an incoming message is processed. RFC 3411 does not define ASIs for the "Send SNMP Request Message to Network", "Receive SNMP Response Message from Network", "Receive SNMP Message from Network" and "Send SNMP message to Network" arrows in these diagrams.

This document defines two ASIs corresponding to these arrows: a sendMessage ASI to send SNMP messages to the network and a receiveMessage ASI to receive SNMP messages from the network. These ASIs are used for all SNMP messages, regardless of pduType.

## 5. Cached Information and References

When performing SNMP processing, there are two levels of state information that might need to be retained: the immediate state linking a request-response pair and a potentially longer-term state relating to transport and security.

The RFC 3411 architecture uses caches to maintain the short-term message state, and uses references in the ASIs to pass this information between subsystems.

This document defines the requirements for a cache to handle additional short-term message state and longer-term transport state information, using a tmStateReference parameter to pass this information between subsystems.

To simplify the elements of procedure, the release of state information is not always explicitly specified. As a general rule, if state information is available when a message being processed gets discarded, the state related to that message should also be discarded. If state information is available when a relationship between engines is severed, such as the closing of a transport session, the state information for that relationship should also be discarded.

Since the contents of a cache are meaningful only within an implementation, and not on-the-wire, the format of the cache is implementation-specific.

### 5.1. securityStateReference

The securityStateReference parameter is defined in RFC 3411. Its primary purpose is to provide a mapping between a request and the corresponding response. This cache is not accessible to Transport Models, and an entry is typically only retained for the lifetime of a request-response pair of messages.

### 5.2. tmStateReference

For each transport session, information about the transport security is stored in a tmState cache or datastore that is referenced by a tmStateReference. The tmStateReference parameter is used to pass model-specific and mechanism-specific parameters between the Transport Subsystem and transport-aware Security Models.

In general, when necessary, the tmState is populated by the Security Model for outgoing messages and by the Transport Model for incoming messages. However, in both cases, the model populating the tmState

might have incomplete information, and the missing information might be populated by the other model when the information becomes available.

The tmState might contain both long-term and short-term information. The session information typically remains valid for the duration of the transport session, might be used for several messages, and might be stored in a local configuration datastore. Some information has a shorter lifespan, such as tmSameSecurity and tmRequestedSecurityLevel, which are associated with a specific message.

Since this cache is only used within an implementation, and not on-the-wire, the precise contents and format of the cache are implementation-dependent. For architectural modularity between Transport Models and transport-aware Security Models, a fully-defined tmState MUST conceptually include at least the following fields:

- tmTransportDomain
- tmTransportAddress
- tmSecurityName
- tmRequestedSecurityLevel
- tmTransportSecurityLevel
- tmSameSecurity
- tmSessionID

The details of these fields are described in the following subsections.

#### 5.2.1. Transport Information

Information about the source of an incoming SNMP message is passed up from the Transport Subsystem as far as the Message Processing Subsystem. However, these parameters are not included in the processIncomingMsg ASI defined in RFC 3411; hence, this information is not directly available to the Security Model.

A transport-aware Security Model might wish to take account of the transport protocol and originating address when authenticating the request and setting up the authorization parameters. It is therefore

necessary for the Transport Model to include this information in the tmStateReference cache so that it is accessible to the Security Model.

- tmTransportDomain: the transport protocol (and hence the Transport Model) used to receive the incoming message.

- tmTransportAddress: the source of the incoming message.

The ASIs used for processing an outgoing message all include explicit transportDomain and transportAddress parameters. The values within the securityStateReference cache might override these parameters for outgoing messages.

#### 5.2.2. securityName

There are actually three distinct "identities" that can be identified during the processing of an SNMP request over a secure transport:

- transport principal: the transport-authenticated identity on whose behalf the secure transport connection was (or should be) established. This value is transport-, mechanism-, and implementation-specific, and is only used within a given Transport Model.

- tmSecurityName: a human-readable name (in snmpAdminString format) representing this transport identity. This value is transport- and implementation-specific, and is only used (directly) by the Transport and Security Models.

- securityName: a human-readable name (in snmpAdminString format) representing the SNMP principal in a model-independent manner. This value is used directly by SNMP Applications, the Access Control Subsystem, the Message Processing Subsystem, and the Security Subsystem.

The transport principal might or might not be the same as the tmSecurityName. Similarly, the tmSecurityName might or might not be the same as the securityName as seen by the Application and Access Control Subsystems. In particular, a non-transport-aware Security Model will ignore tmSecurityName completely when determining the SNMP securityName.

However, it is important that the mapping between the transport principal and the SNMP securityName (for transport-aware Security Models) is consistent and predictable in order to allow configuration of suitable access control and the establishment of transport connections.

#### 5.2.3. securityLevel

There are two distinct issues relating to security level as applied to secure transports. For clarity, these are handled by separate fields in the tmStateReference cache:

- tmTransportSecurityLevel: an indication from the Transport Model of the level of security offered by this session. The Security Model can use this to ensure that incoming messages were suitably protected before acting on them.

- tmRequestedSecurityLevel: an indication from the Security Model of the level of security required to be provided by the transport protocol. The Transport Model can use this to ensure that outgoing messages will not be sent over an insufficiently secure session.

#### 5.2.4. Session Information

For security reasons, if a secure transport session is closed between the time a request message is received and the corresponding response message is sent, then the response message SHOULD be discarded, even if a new session has been established. The SNMPv3 WG decided that this should be a "SHOULD" architecturally, and it is a Security-Model-specific decision whether to REQUIRE this.

- tmSameSecurity: this flag is used by a transport-aware Security Model to indicate whether the Transport Model MUST enforce this restriction.

- tmSessionID: in order to verify whether the session has changed, the Transport Model must be able to compare the session used to receive the original request with the one to be used to send the response. This typically needs some form of session identifier. This value is only ever used by the Transport Model, so the format and interpretation of this field are model-specific and implementation-dependent.

When processing an outgoing message, if tmSameSecurity is true, then the tmSessionID MUST match the current transport session; otherwise, the message MUST be discarded and the Dispatcher notified that sending the message failed.

## 6. Abstract Service Interfaces

Abstract service interfaces have been defined by RFC 3411 to describe the conceptual data flows between the various subsystems within an SNMP entity and to help keep the subsystems independent of each other except for the common parameters.

This document introduces a couple of new ASIs to define the interface between the Transport and Dispatcher Subsystems; it also extends some of the ASIs defined in RFC 3411 to include transport-related information.

This document follows the example of RFC 3411 regarding the release of state information and regarding error indications.

1) The release of state information is not always explicitly specified in a Transport Model. As a general rule, if state information is available when a message gets discarded, the message-state information should also be released, and if state information is available when a session is closed, the session-state information should also be released. Keeping sensitive security information longer than necessary might introduce potential vulnerabilities to an implementation.

2)An error indication in statusInformation will typically include the Object Identifier (OID) and value for an incremented error counter. This might be accompanied by values for contextEngineID and contextName for this counter, a value for securityLevel, and the appropriate state reference if the information is available at the point where the error is detected.

### 6.1. sendMessage ASI

The sendMessage ASI is used to pass a message from the Dispatcher to the appropriate Transport Model for sending. The sendMessageASI defined in this document replaces the text "Send SNMP Request Message to Network" that appears in the diagram in Section 4.6.1 of RFC 3411 and the text "Send SNMP Message to Network" that appears in Section 4.6.2 of RFC 3411.

If present and valid, the tmStateReference refers to a cache containing Transport-Model-specific parameters for the transport and transport security. How a tmStateReference is determined to be present and valid is implementation-dependent. How the information in the cache is used is Transport-Model-dependent and implementation-dependent.

This might sound underspecified, but a Transport Model might be something like SNMP over UDP over IPv6, where no security is provided, so it might have no mechanisms for utilizing a tmStateReference cache.

```
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             -- reference to transport state
 )
```

### 6.2. Changes to RFC 3411 Outgoing ASIs

Additional parameters have been added to the ASIs defined in RFC 3411 that are concerned with communication between the Dispatcher and Message Processing Subsystems, and between the Message Processing and Security Subsystems.

#### 6.2.1. Message Processing Subsystem Primitives

A tmStateReference parameter has been added as an OUT parameter to the prepareOutgoingMessage and prepareResponseMessage ASIs. This is passed from the Message Processing Subsystem to the Dispatcher, and from there to the Transport Subsystem.

How or if the Message Processing Subsystem modifies or utilizes the contents of the cache is Message-Processing-Model specific.

```
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
   OUT  tmStateReference        -- (NEW) reference to transport state
              )

   statusInformation =          -- success or errorIndication
   prepareResponseMessage(
   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  maxSizeResponseScopedPDU -- maximum size able to accept
   IN  stateReference           -- reference to state information
                                -- as presented with the request
   IN  statusInformation        -- success or errorIndication
                                -- error counter OID/value if error
   OUT destTransportDomain      -- destination transport domain
   OUT destTransportAddress     -- destination transport address
   OUT outgoingMessage          -- the message to send
   OUT outgoingMessageLength    -- its length
   OUT tmStateReference         -- (NEW) reference to transport state
              )
```

### 6.2.2. Security Subsystem Primitives

transportDomain and transportAddress parameters have been added as IN parameters to the generateRequestMsg and generateResponseMsg ASIs, and a tmStateReference parameter has been added as an OUT parameter. The transportDomain and transportAddress parameters will have been passed into the Message Processing Subsystem from the Dispatcher and are passed on to the Security Subsystem. The tmStateReference parameter will be passed from the Security Subsystem back to the Message Processing Subsystem, and on to the Dispatcher and Transport Subsystems.

If a cache exists for a session identifiable from the tmTransportDomain, tmTransportAddress, tmSecurityName, and requested securityLevel, then a transport-aware Security Model might create a tmStateReference parameter to this cache and pass that as an OUT parameter.

```
   statusInformation =
   generateRequestMsg(
     IN   transportDomain         -- (NEW) destination transport domain
     IN   transportAddress        -- (NEW) destination transport address
     IN   messageProcessingModel  -- typically, SNMP version
     IN   globalData              -- message header, admin data
     IN   maxMessageSize          -- of the sending SNMP entity
     IN   securityModel           -- for the outgoing message
     IN   securityEngineID        -- authoritative SNMP entity
     IN   securityName            -- on behalf of this principal
     IN   securityLevel           -- Level of Security requested
     IN   scopedPDU               -- message (plaintext) payload
     OUT  securityParameters      -- filled in by Security Module
     OUT  wholeMsg                -- complete generated message
     OUT  wholeMsgLength          -- length of generated message
     OUT  tmStateReference        -- (NEW) reference to transport state
             )

   statusInformation =
   generateResponseMsg(
     IN   transportDomain         -- (NEW) destination transport domain
     IN   transportAddress        -- (NEW) destination transport address
     IN   messageProcessingModel -- Message Processing Model
     IN   globalData             -- msgGlobalData
     IN   maxMessageSize         -- from msgMaxSize
     IN   securityModel          -- as determined by MPM
     IN   securityEngineID       -- the value of snmpEngineID
     IN   securityName           -- on behalf of this principal
     IN   securityLevel          -- for the outgoing message
     IN   scopedPDU              -- as provided by MPM
     IN   securityStateReference -- as provided by MPM
     OUT  securityParameters     -- filled in by Security Module
     OUT  wholeMsg               -- complete generated message
     OUT  wholeMsgLength         -- length of generated message
     OUT  tmStateReference       -- (NEW) reference to transport state
             )
```

### 6.3. The receiveMessage ASI

The receiveMessage ASI is used to pass a message from the Transport Subsystem to the Dispatcher. The receiveMessage ASI replaces the text "Receive SNMP Response Message from Network" that appears in the diagram in Section 4.6.1 of RFC 3411 and the text "Receive SNMP Message from Network" from Section 4.6.2 of RFC3411.

When a message is received on a given transport session, if a cache does not already exist for that session, the Transport Model might create one, referenced by tmStateReference. The contents of this

cache are discussed in Section 5. How this information is determined is implementation- and Transport-Model-specific.

"Might create one" might sound underspecified, but a Transport Model might be something like SNMP over UDP over IPv6, where transport security is not provided, so it might not create a cache.

The Transport Model does not know the securityModel for an incoming message; this will be determined by the Message Processing Model in a Message-Processing-Model-dependent manner.

```
statusInformation =
receiveMessage(
IN   transportDomain              -- origin transport domain
IN   transportAddress             -- origin transport address
IN   incomingMessage              -- the message received
IN   incomingMessageLength        -- its length
IN   tmStateReference             -- reference to transport state
 )
```

### 6.4. Changes to RFC 3411 Incoming ASIs

The tmStateReference parameter has also been added to some of the incoming ASIs defined in RFC 3411. How or if a Message Processing Model or Security Model uses tmStateReference is message-processing- and Security-Model-specific.

This might sound underspecified, but a Message Processing Model might have access to all the information from the cache and from the message. The Message Processing Model might determine that the USM Security Model is specified in an SNMPv3 message header; the USM Security Model has no need of values in the tmStateReference cache to authenticate and secure the SNMP message, but an Application might have specified to use a secure transport such as that provided by the SSH Transport Model to send the message to its destination.

#### 6.4.1. Message Processing Subsystem Primitive

The tmStateReference parameter of prepareDataElements is passed from the Dispatcher to the Message Processing Subsystem. How or if the Message Processing Subsystem modifies or utilizes the contents of the cache is Message-Processing-Model-specific.

```
result =                      -- SUCCESS or errorIndication
prepareDataElements(
IN   transportDomain          -- origin transport domain
IN   transportAddress         -- origin transport address
IN   wholeMsg                 -- as received from the network
```

```
   IN   wholeMsgLength            -- as received from the network
   IN   tmStateReference          -- (NEW) from the Transport Model
   OUT  messageProcessingModel    -- typically, SNMP version
   OUT  securityModel             -- Security Model to use
   OUT  securityName              -- on behalf of this principal
   OUT  securityLevel             -- Level of Security requested
   OUT  contextEngineID           -- data from/at this entity
   OUT  contextName               -- data from/in this context
   OUT  pduVersion                -- the version of the PDU
   OUT  PDU                       -- SNMP Protocol Data Unit
   OUT  pduType                   -- SNMP PDU type
   OUT  sendPduHandle             -- handle for matched request
   OUT  maxSizeResponseScopedPDU  -- maximum size sender can accept
   OUT  statusInformation         -- success or errorIndication
                                  -- error counter OID/value if error
   OUT  stateReference            -- reference to state information
                                  -- to be used for possible Response
   )
```

### 6.4.2. Security Subsystem Primitive

The processIncomingMessage ASI passes tmStateReference from the Message Processing Subsystem to the Security Subsystem.

If tmStateReference is present and valid, an appropriate Security Model might utilize the information in the cache. How or if the Security Subsystem utilizes the information in the cache is Security-Model-specific.

```
   statusInformation =  -- errorIndication or success
                            -- error counter OID/value if error
   processIncomingMsg(
   IN   messageProcessingModel    -- typically, SNMP version
   IN   maxMessageSize            -- of the sending SNMP entity
   IN   securityParameters        -- for the received message
   IN   securityModel             -- for the received message
   IN   securityLevel             -- Level of Security
   IN   wholeMsg                  -- as received on the wire
   IN   wholeMsgLength            -- length as received on the wire
   IN   tmStateReference          -- (NEW) from the Transport Model
   OUT  securityEngineID          -- authoritative SNMP entity
   OUT  securityName              -- identification of the principal
   OUT  scopedPDU,                -- message (plaintext) payload
   OUT  maxSizeResponseScopedPDU  -- maximum size sender can handle
   OUT  securityStateReference    -- reference to security state
                                  -- information, needed for response
   )
```

## 7. Security Considerations

This document defines an architectural approach that permits SNMP to utilize transport-layer security services. Each proposed Transport Model should discuss the security considerations of that Transport Model.

It is considered desirable by some industry segments that SNMP Transport Models 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. Each proposed Transport Model should include a discussion in its security considerations of whether perfect forward secrecy is appropriate for that Transport Model.

The denial-of-service characteristics of various Transport Models and security protocols will vary and should be evaluated when determining the applicability of a Transport Model to a particular deployment situation.

Since the cache will contain security-related parameters, implementers SHOULD store this information (in memory or in persistent storage) in a manner to protect it from unauthorized disclosure and/or modification.

Care must be taken to ensure that an SNMP engine is sending packets out over a transport using credentials that are legal for that engine to use on behalf of that user. Otherwise, an engine that has multiple transports open might be "tricked" into sending a message through the wrong transport.

A Security Model might have multiple sources from which to define the securityName and securityLevel. The use of a secure Transport Model does not imply that the securityName and securityLevel chosen by the Security Model represent the transport-authenticated identity or the transport-provided security services. The securityModel, securityName, and securityLevel parameters are a related set, and an administrator should understand how the specified securityModel selects the corresponding securityName and securityLevel.

### 7.1. Coexistence, Security Parameters, and Access Control

In the RFC 3411 architecture, the Message Processing Model makes the decision about which Security Model to use. The architectural change described by this document does not alter that.

The architecture change described by this document does, however, allow SNMP to support two different approaches to security -- message-driven security and transport-driven security. With message-driven security, SNMP provides its own security and passes security parameters within the SNMP message; with transport-driven security, SNMP depends on an external entity to provide security during transport by "wrapping" the SNMP message.

Using a non-transport-aware Security Model with a secure Transport Model is NOT RECOMMENDED for the following reasons.

Security Models defined before the Transport Security Model (i.e., SNMPv1, SNMPv2c, and USM) do not support transport-based security and only have access to the security parameters contained within the SNMP message. They do not know about the security parameters associated with a secure transport. As a result, the Access Control Subsystem bases its decisions on the security parameters extracted from the SNMP message, not on transport-based security parameters.

Implications of combining older Security Models with Secure Transport Models are known. The securityName used for access control decisions is based on the message-driven identity, which might be unauthenticated, and not on the transport-driven, authenticated identity:

- An SNMPv1 message will always be paired with an SNMPv1 Security Model (per RFC 3584), regardless of the transport mapping or Transport Model used, and access controls will be based on the unauthenticated community name.

- An SNMPv2c message will always be paired with an SNMPv2c Security Model (per RFC 3584), regardless of the transport mapping or Transport Model used, and access controls will be based on the unauthenticated community name.

- An SNMPv3 message will always be paired with the securityModel specified in the msgSecurityParameters field of the message (per RFC 3412), regardless of the transport mapping or Transport Model used. If the SNMPv3 message specifies the User-based Security Model (USM) with noAuthNoPriv, then the access controls will be based on the unauthenticated USM user.

- For outgoing messages, if a Secure Transport Model is selected in combination with a Security Model that does not populate a tmStateReference, the Secure Transport Model SHOULD detect the lack of a valid tmStateReference and fail.

In times of network stress, a Secure Transport Model might not work properly if its underlying security mechanisms (e.g., Network Time Protocol (NTP) or Authentication, Authorization, and Accounting (AAA) protocols or certificate authorities) are not reachable. The User-based Security Model was explicitly designed to not depend upon external network services, and provides its own security services. It is RECOMMENDED that operators provision authPriv USM as a fallback mechanism to supplement any Security Model or Transport Model that has external dependencies, so that secure SNMP communications can continue when the external network service is not available.

## 8. IANA Considerations

IANA has created a new registry in the Simple Network Management Protocol (SNMP) Number Spaces. The new registry is called "SNMP Transport Domains". This registry contains US-ASCII alpha-numeric strings of one to four characters to identify prefixes for corresponding SNMP transport domains. Each transport domain MUST have an OID assignment under snmpDomains [RFC2578]. Values are to be assigned via [RFC5226] "Specification Required".

The registry has been populated with the following initial entries:

Registry Name: SNMP Transport Domains
Reference: [RFC2578] [RFC3417] [RFC5590]
Registration Procedures: Specification Required
Each domain is assigned a MIB-defined OID under snmpDomains

| Prefix | snmpDomains | Reference |
|---|---|---|
| udp | snmpUDPDomain | [RFC3417] [RFC5590] |
| clns | snmpCLNSDomain | [RFC3417] [RFC5590] |
| cons | snmpCONSDomain | [RFC3417] [RFC5590] |
| ddp | snmpDDPDomain | [RFC3417] [RFC5590] |
| ipx | snmpIPXDomain | [RFC3417] [RFC5590] |
| prxy | rfc1157Domain | [RFC3417] [RFC5590] |

## 9. 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, David Harrington, Keith McCloghrie, Kaushik Narayan, David 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 performed detailed reviews: Wes Hardaker, Jeffrey Hutzelman, Tom Petch, Dave Shield, and Bert Wijnen.

## 10. References

### 10.1. Normative References

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

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

[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.

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

### 10.2. Informative References

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

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

[RFC3584] Frye, R., Levi, D., Routhier, S., and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", BCP 74, RFC 3584, August 2003.

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

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, March 2009.

[RFC5591] Harrington, D. and W. Hardaker, "Transport Security Model for the Simple Network Management Protocol (SNMP)", RFC 5591, June 2009.

[RFC5592] Harrington, D., Salowey, J., and W. Hardaker, "Secure Shell Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5592, June 2009.

## Appendix A. Why tmStateReference?

This appendix considers why a cache-based approach was selected for passing parameters.

There are four approaches that could be used for passing information between the Transport Model and a Security Model.

1. One could define an ASI to supplement the existing ASIs.

2. One could add a header to encapsulate the SNMP message.

3. One could utilize fields already defined in the existing SNMPv3 message.

4. One could pass the information in an implementation-specific cache or via a MIB module.

### A.1. Define an Abstract Service Interface

Abstract Service Interfaces (ASIs) 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 Subsystem to the Security Subsystem has the advantage of being consistent with existing RFC 3411/3412 practice; it also helps to ensure that any Transport Model proposals pass the necessary data and do not cause side effects by creating model-specific dependencies between itself and models or subsystems other than those that are clearly defined by an ASI.

### A.2. Using an Encapsulating Header

A header could encapsulate the SNMP message to pass necessary information from the Transport Model to the Dispatcher and then to a Message Processing Model. The message header would be included in the wholeMessage ASI parameter and would be removed by a corresponding Message Processing Model. This would imply the (one and only) Message 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 Security Model.

### A.3. Modifying Existing Fields in an SNMP Message

[RFC3412] defines the SNMPv3 message, which contains fields to pass security-related parameters. The Transport Subsystem could use these fields in an SNMPv3 message (or comparable fields in other message

formats) to pass information between Transport Models in different SNMP engines and to pass information between a Transport Model and a corresponding Message Processing Model.

If the fields in an incoming SNMPv3 message are changed by the Transport Model before passing it to the Security Model, then the Transport Model 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 in order for the Transport Model to know which fields could be modified. This would seriously violate the modularity of the architecture.

### A.4. Using a Cache

This document describes a cache into which the Transport Model (TM) puts information about the security applied to an incoming message; a Security Model can extract that information from the cache. Given that there might be multiple TM security caches, a tmStateReference is passed as an extra parameter in the ASIs between the Transport Subsystem and the Security Subsystem so that the Security Model knows which cache of information to consult.

This approach does create dependencies between a specific Transport Model and a corresponding specific Security Model. However, the approach of passing a model-independent reference to a model-dependent cache is consistent with the securityStateReference already being passed around in the RFC 3411 ASIs.

## Authors' Addresses

David Harrington
Huawei Technologies (USA)
1700 Alma Dr. Suite 100
Plano, TX 75075
USA

Phone: +1 603 436 8634
EMail: ietfdbh@comcast.net

Juergen Schoenwaelder
Jacobs University Bremen
Campus Ring 1
28725 Bremen
Germany

Phone: +49 421 200-3587
EMail: j.schoenwaelder@jacobs-university.de

Network Working Group
Request for Comments: 5591
Category: Standards Track

D. Harrington
Huawei Technologies (USA)
W. Hardaker
Cobham Analytic Solutions

June 2009

# Transport Security Model for the Simple Network Management Protocol (SNMP)

## Status of This Memo



## Copyright Notice

Abstract

This memo describes a Transport Security Model for the Simple Network Management Protocol (SNMP).

This memo also defines a portion of the Management Information Base (MIB) for monitoring and managing the Transport Security Model for SNMP.

Table of Contents

## 1. Introduction

This memo describes a Transport Security Model for the Simple Network Management Protocol for use with secure Transport Models in the Transport Subsystem [RFC5590].

This memo also defines a portion of the Management Information Base (MIB) for monitoring and managing the Transport Security Model for SNMP.

It is important to understand the SNMP architecture and the terminology of the architecture to understand where the Transport Security Model described in this memo fits into the architecture and interacts with other subsystems and models within the architecture. It is expected that readers will have also read and understood [RFC3411], [RFC3412], [RFC3413], and [RFC3418].

### 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 [RFC2119].

Lowercase versions of the keywords should be read as in normal English. They will usually, but not always, be used in a context that relates to compatibility with the RFC 3411 architecture or the subsystem defined here but that might have no impact on on-the-wire compatibility. These terms are used as guidance for designers of proposed IETF models to make the designs compatible with RFC 3411 subsystems and Abstract Service Interfaces (ASIs). Implementers are free to implement differently. Some usages of these lowercase terms are simply normal English usage.

For consistency with SNMP-related specifications, this document favors terminology as defined in STD 62, rather than favoring terminology that is consistent with non-SNMP specifications that use different variations of the same terminology. This is consistent with the IESG decision to not require the SNMPv3 terminology be modified to match the usage of other non-SNMP specifications when SNMPv3 was advanced to Full Standard.

Authentication in this document typically refers to the English meaning of "serving to prove the authenticity of" the message, not data source authentication or peer identity authentication.

The terms "manager" and "agent" are not used in this document because, in the RFC 3411 architecture, all SNMP entities have the capability of acting as manager, agent, or both depending on the SNMP applications included in the engine. Where distinction is needed, the application names of command generator, command responder, notification originator, notification receiver, and proxy forwarder are used. See "Simple Network Management Protocol (SNMP) Applications" [RFC3413] for further information.

While security protocols frequently refer to a user, the terminology used in [RFC3411] and in this memo is "principal". A 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 each acting in a particular role, an application or a set of applications, or a combination of these within an administrative domain.

### 1.3. Modularity

The reader is expected to have read and understood the description of the SNMP architecture, as defined in [RFC3411], and the architecture extension specified in "Transport Subsystem for the Simple Network Management Protocol (SNMP)" [RFC5590], which enables the use of external "lower-layer transport" protocols to provide message

security. Transport Models are tied into the SNMP architecture through the Transport Subsystem. The Transport Security Model is designed to work with such lower-layer, secure Transport Models.

In keeping with the RFC 3411 design decisions to use self-contained documents, this memo includes the elements of procedure plus associated MIB objects that are needed for processing the Transport Security Model for SNMP. These MIB objects SHOULD NOT be referenced in other documents. This allows the Transport Security Model to be designed and documented as independent and self-contained, having no direct impact on other modules. It also allows this module to be upgraded and supplemented as the need arises, and to move along the standards track on different time-lines from other modules.

This modularity of specification is not meant to be interpreted as imposing any specific requirements on implementation.

### 1.4. Motivation

This memo describes a Security Model to make use of Transport Models that use lower-layer, secure transports and existing and commonly deployed security infrastructures. This Security Model is designed to meet the security and operational needs of network administrators, maximize usability in operational environments to achieve high deployment success, and at the same time minimize implementation and deployment costs to minimize the time until deployment is possible.

### 1.5. Constraints

The design of this SNMP Security Model is also influenced by the following constraints:

1. In times of network stress, the security protocol and its underlying security mechanisms SHOULD NOT depend solely upon the ready availability of other network services (e.g., Network Time Protocol (NTP) or Authentication, Authorization, and Accounting (AAA) protocols).

2. When the network is not under stress, the Security Model and its underlying security mechanisms MAY depend upon the ready availability of other network services.

3. It might not be possible for the Security Model to determine when the network is under stress.

4. A Security Model SHOULD NOT require changes to the SNMP architecture.

5. A Security Model SHOULD NOT require changes to the underlying security protocol.

## 2. How the Transport Security Model Fits in the Architecture

The Transport Security Model is designed to fit into the RFC 3411 architecture as a Security Model in the Security Subsystem and to utilize the services of a secure Transport Model.

For incoming messages, a secure Transport Model will pass a tmStateReference cache, described in [RFC5590]. To maintain RFC 3411 modularity, the Transport Model will not know which securityModel will process the incoming message; the Message Processing Model will determine this. If the Transport Security Model is used with a non-secure Transport Model, then the cache will not exist or will not be populated with security parameters, which will cause the Transport Security Model to return an error (see Section 5.2).

The Transport Security Model will create the securityName and securityLevel to be passed to applications, and will verify that the tmTransportSecurityLevel reported by the Transport Model is at least as strong as the securityLevel requested by the Message Processing Model.

For outgoing messages, the Transport Security Model will create a tmStateReference cache (or use an existing one), and will pass the tmStateReference to the specified Transport Model.

### 2.1. Security Capabilities of this Model

#### 2.1.1. Threats

The Transport Security Model is compatible with the RFC 3411 architecture and provides protection against the threats identified by the RFC 3411 architecture. However, the Transport Security Model does not provide security mechanisms such as authentication and encryption itself. Which threats are addressed and how they are mitigated depends on the Transport Model used. To avoid creating potential security vulnerabilities, operators should configure their system so this Security Model is always used with a Transport Model that provides appropriate security, where "appropriate" for a particular deployment is an administrative decision.

#### 2.1.2. Security Levels

The RFC 3411 architecture recognizes three levels of security:

- without authentication and without privacy (noAuthNoPriv)
- with authentication but without privacy (authNoPriv)
- with authentication and with privacy (authPriv)

The model-independent securityLevel parameter is used to request specific levels of security for outgoing messages and to assert that specific levels of security were applied during the transport and processing of incoming messages.

The transport-layer algorithms used to provide security should not be exposed to the Transport Security Model, as the Transport Security Model has no mechanisms by which it can test whether an assertion made by a Transport Model is accurate.

The Transport Security Model trusts that the underlying secure transport connection has been properly configured to support security characteristics at least as strong as reported in tmTransportSecurityLevel.

### 2.2. Transport Sessions

The Transport Security Model does not work with transport sessions directly. Instead the transport-related state is associated with a unique combination of transportDomain, transportAddress, securityName, and securityLevel, and is referenced via the tmStateReference parameter. How and if this is mapped to a particular transport or channel is the responsibility of the Transport Subsystem.

### 2.3. Coexistence

In the RFC 3411 architecture, a Message Processing Model determines which Security Model SHALL be called. As of this writing, IANA has registered four Message Processing Models (SNMPv1, SNMPv2c, SNMPv2u/SNMPv2*, and SNMPv3) and three other Security Models (SNMPv1, SNMPv2c, and the User-based Security Model).

#### 2.3.1. Coexistence with Message Processing Models

The SNMPv1 and SNMPv2c message processing described in BCP 74 [RFC3584] always selects the SNMPv1(1) and SNMPv2c(2) Security Models. Since there is no mechanism defined in RFC 3584 to select an

alternative Security Model, SNMPv1 and SNMPv2c messages cannot use the Transport Security Model. Messages might still be able to be conveyed over a secure transport protocol, but the Transport Security Model will not be invoked.

The SNMPv2u/SNMPv2* Message Processing Model is an historic artifact for which there is no existing IETF specification.

The SNMPv3 message processing defined in [RFC3412] extracts the securityModel from the msgSecurityModel field of an incoming SNMPv3Message. When this value is transportSecurityModel(4), security processing is directed to the Transport Security Model. For an outgoing message to be secured using the Transport Security Model, the application MUST specify a securityModel parameter value of transportSecurityModel(4) in the sendPdu Abstract Service Interface (ASI).

#### 2.3.2. Coexistence with Other Security Models

The Transport Security Model uses its own MIB module for processing to maintain independence from other Security Models. This allows the Transport Security Model to coexist with other Security Models, such as the User-based Security Model (USM) [RFC3414].

#### 2.3.3. Coexistence with Transport Models

The Transport Security Model (TSM) MAY work with multiple Transport Models, but the RFC 3411 Abstract Service Interfaces (ASIs) do not carry a value for the Transport Model. The MIB module defined in this memo allows an administrator to configure whether or not TSM prepends a Transport Model prefix to the securityName. This will allow SNMP applications to consider Transport Model as a factor when making decisions, such as access control, notification generation, and proxy forwarding.

To have SNMP properly utilize the security services coordinated by the Transport Security Model, this Security Model MUST only be used with Transport Models that know how to process a tmStateReference, such as the Secure Shell Transport Model [RFC5592].

## 3. Cached Information and References

When performing SNMP processing, there are two levels of state information that might need to be retained: the immediate state linking a request-response pair and a potentially longer-term state relating to transport and security. "Transport Subsystem for the Simple Network Management Protocol (SNMP)" [RFC5590] defines general requirements for caches and references.

This document defines additional cache requirements related to the Transport Security Model.

### 3.1. Transport Security Model Cached Information

The Transport Security Model has specific responsibilities regarding the cached information.

#### 3.1.1. securityStateReference

The Transport Security Model adds the tmStateReference received from the processIncomingMsg ASI to the securityStateReference. This tmStateReference can then be retrieved during the generateResponseMsg ASI so that it can be passed back to the Transport Model.

#### 3.1.2. tmStateReference

For outgoing messages, the Transport Security Model uses parameters provided by the SNMP application to look up or create a tmStateReference.

For the Transport Security Model, the security parameters used for a response MUST be the same as those used for the corresponding request. This Security Model uses the tmStateReference stored as part of the securityStateReference when appropriate. For responses and reports, this Security Model sets the tmSameSecurity flag to true in the tmStateReference before passing it to a Transport Model.

For incoming messages, the Transport Security Model uses parameters provided in the tmStateReference cache to establish a securityName, and to verify adequate security levels.

#### 3.1.3. Prefixes and securityNames

The SNMP-VIEW-BASED-ACM-MIB module [RFC3415], the SNMP-TARGET-MIB module [RFC3413], and other MIB modules contain objects to configure security parameters for use by applications such as access control, notification generation, and proxy forwarding.

Transport domains and their corresponding prefixes are coordinated via the IANA registry "SNMP Transport Domains".

If snmpTsmConfigurationUsePrefix is set to true, then all securityNames provided by, or provided to, the Transport Security Model MUST include a valid transport domain prefix.

If snmpTsmConfigurationUsePrefix is set to false, then all securityNames provided by, or provided to, the Transport Security Model MUST NOT include a transport domain prefix.

The tmSecurityName in the tmStateReference stored as part of the securityStateReference does not contain a prefix.

## 4. Processing an Outgoing Message

An error indication might return an Object Identifier (OID) and value for an incremented counter, a value for securityLevel, values for contextEngineID and contextName for the counter, and the securityStateReference, if this information is available at the point where the error is detected.

### 4.1. Security Processing for an Outgoing Message

This section describes the procedure followed by the Transport Security Model.

The parameters needed for generating a message are supplied to the Security Model by the Message Processing Model via the generateRequestMsg() or the generateResponseMsg() ASI. The Transport Subsystem architectural extension has added the transportDomain, transportAddress, and tmStateReference parameters to the original RFC 3411 ASIs.

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

  statusInformation = -- success or errorIndication
       generateResponseMsg(
       IN   messageProcessingModel  -- typically, SNMP version
```

```
        IN   globalData              -- message header, admin data
        IN   maxMessageSize          -- of the sending SNMP entity
        IN   transportDomain         -- (NEW) specified by application
        IN   transportAddress        -- (NEW) specified by application
        IN   securityModel           -- for the outgoing message
        IN   securityEngineID        -- authoritative SNMP entity
        IN   securityName            -- on behalf of this principal
        IN   securityLevel           -- Level of Security requested
        IN   scopedPDU               -- message (plaintext) payload
        IN   securityStateReference  -- reference to security state
                                     -- information from original
                                     -- request
        OUT  securityParameters      -- filled in by Security Module
        OUT  wholeMsg                -- complete generated message
        OUT  wholeMsgLength          -- length of generated message
        OUT  tmStateReference        -- (NEW) transport info
             )
```

## 4.2. Elements of Procedure for Outgoing Messages

1. If there is a securityStateReference (Response or Report message), then this Security Model uses the cached information rather than the information provided by the ASI. Extract the tmStateReference from the securityStateReference cache. Set the tmRequestedSecurityLevel to the value of the extracted tmTransportSecurityLevel. Set the tmSameSecurity parameter in the tmStateReference cache to true. The cachedSecurityData for this message can now be discarded.

2. If there is no securityStateReference (e.g., a Request-type or Notification message), then create a tmStateReference cache. Set tmTransportDomain to the value of transportDomain, tmTransportAddress to the value of transportAddress, and tmRequestedSecurityLevel to the value of securityLevel. (Implementers might optimize by pointing to saved copies of these session-specific values.) Set the transaction-specific tmSameSecurity parameter to false.

   If the snmpTsmConfigurationUsePrefix object is set to false, then set tmSecurityName to the value of securityName.

   If the snmpTsmConfigurationUsePrefix object is set to true, then use the transportDomain to look up the corresponding prefix. (Since the securityStateReference stores the tmStateReference with the tmSecurityName for the incoming message, and since tmSecurityName never has a prefix, the prefix-stripping step only occurs when we are not using the securityStateReference).

If the prefix lookup fails for any reason, then the snmpTsmUnknownPrefixes counter is incremented, an error indication is returned to the calling module, and message processing stops.

If the lookup succeeds, but there is no prefix in the securityName, or the prefix returned does not match the prefix in the securityName, or the length of the prefix is less than 1 or greater than 4 US-ASCII alpha-numeric characters, then the snmpTsmInvalidPrefixes counter is incremented, an error indication is returned to the calling module, and message processing stops.

Strip the transport-specific prefix and trailing ':' character (US-ASCII 0x3a) from the securityName. Set tmSecurityName to the value of securityName.

3. Set securityParameters to a zero-length OCTET STRING ('0400').

4. Combine the message parts into a wholeMsg and calculate wholeMsgLength.

5. The wholeMsg, wholeMsgLength, securityParameters, and tmStateReference are returned to the calling Message Processing Model with the statusInformation set to success.

## 5. Processing an Incoming SNMP Message

An error indication might return an OID and value for an incremented counter, a value for securityLevel, values for contextEngineID and contextName for the counter, and the securityStateReference, if this information is available at the point where the error is detected.

### 5.1. Security Processing for an Incoming Message

This section describes the procedure followed by the Transport Security Model whenever it receives an incoming message from a Message Processing Model. The ASI from a Message Processing Model to the Security Subsystem for a received message is:

```
statusInformation =  -- errorIndication or success
                         -- error counter OID/value if error
processIncomingMsg(
IN   messageProcessingModel    -- typically, SNMP version
IN   maxMessageSize            -- from the received message
IN   securityParameters        -- from the received message
IN   securityModel             -- from the received message
IN   securityLevel             -- from the received message
```

```
   IN   wholeMsg                  -- as received on the wire
   IN   wholeMsgLength            -- length as received on the wire
   IN   tmStateReference          -- (NEW) from the Transport Model
   OUT  securityEngineID          -- authoritative SNMP entity
   OUT  securityName              -- identification of the principal
   OUT  scopedPDU,                -- message (plaintext) payload
   OUT  maxSizeResponseScopedPDU  -- maximum size sender can handle
   OUT  securityStateReference    -- reference to security state
    )                          -- information, needed for response
```

## 5.2. Elements of Procedure for Incoming Messages

1. Set the securityEngineID to the local snmpEngineID.

2. If tmStateReference does not refer to a cache containing values for tmTransportDomain, tmTransportAddress, tmSecurityName, and tmTransportSecurityLevel, then the snmpTsmInvalidCaches counter is incremented, an error indication is returned to the calling module, and Security Model processing stops for this message.

3. Copy the tmSecurityName to securityName.

   If the snmpTsmConfigurationUsePrefix object is set to true, then use the tmTransportDomain to look up the corresponding prefix.

      If the prefix lookup fails for any reason, then the snmpTsmUnknownPrefixes counter is incremented, an error indication is returned to the calling module, and message processing stops.

      If the lookup succeeds but the prefix length is less than 1 or greater than 4 octets, then the snmpTsmInvalidPrefixes counter is incremented, an error indication is returned to the calling module, and message processing stops.

      Set the securityName to be the concatenation of the prefix, a ':' character (US-ASCII 0x3a), and the tmSecurityName.

4. Compare the value of tmTransportSecurityLevel in the tmStateReference cache to the value of the securityLevel parameter passed in the processIncomingMsg ASI. If securityLevel specifies privacy (Priv) and tmTransportSecurityLevel specifies no privacy (noPriv), or if securityLevel specifies authentication (auth) and tmTransportSecurityLevel specifies no authentication (noAuth) was provided by the Transport Model, then the snmpTsmInadequateSecurityLevels counter is incremented, an error indication (unsupportedSecurityLevel) together with the OID and

value of the incremented counter is returned to the calling module, and Transport Security Model processing stops for this message.

5. The tmStateReference is cached as cachedSecurityData so that a possible response to this message will use the same security parameters. Then securityStateReference is set for subsequent references to this cached data.

6. The scopedPDU component is extracted from the wholeMsg.

7. The maxSizeResponseScopedPDU is calculated. This is the maximum size allowed for a scopedPDU for a possible Response message.

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

## 6. MIB Module Overview

This MIB module provides objects for use only by the Transport Security Model. It defines a configuration scalar and related error counters.

### 6.1. 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.

#### 6.1.1. The snmpTsmStats Subtree

This subtree contains error counters specific to the Transport Security Model.

#### 6.1.2. The snmpTsmConfiguration Subtree

This subtree contains a configuration object that enables administrators to specify if they want a transport domain prefix prepended to securityNames for use by applications.

### 6.2. Relationship to Other MIB Modules

Some management objects defined in other MIB modules are applicable to an entity implementing the Transport Security Model. In particular, it is assumed that an entity implementing the Transport Security Model will implement the SNMP-FRAMEWORK-MIB [RFC3411], the

SNMP-TARGET-MIB [RFC3413], the SNMP-VIEW-BASED-ACM-MIB [RFC3415], and the SNMPv2-MIB [RFC3418]. These are not needed to implement the SNMP-TSM-MIB.

### 6.2.1. MIB Modules Required for IMPORTS

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

## 7. MIB Module Definition

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

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    mib-2, Counter32
      FROM SNMPv2-SMI -- RFC2578
    MODULE-COMPLIANCE, OBJECT-GROUP
      FROM SNMPv2-CONF -- RFC2580
    TruthValue
       FROM SNMPv2-TC -- RFC2579
    ;

snmpTsmMIB MODULE-IDENTITY
    LAST-UPDATED "200906090000Z"
    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
                    Jacobs University Bremen
                    Campus Ring 1
                    28725 Bremen
                    Germany
                    +49 421 200-3587
                    j.schoenwaelder@jacobs-university.de
```

```
                  Editor:
                    David Harrington
                    Huawei Technologies USA
                    1700 Alma Dr.
                    Plano TX 75075
                    USA
                    +1 603-436-8634
                    ietfdbh@comcast.net

                    Wes Hardaker
                    Cobham Analytic Solutions
                    P.O. Box 382
                    Davis, CA  95617
                    USA
                    +1 530 792 1913
                    ietf@hardakers.net
                  "
    DESCRIPTION
       "The Transport Security Model MIB.

        In keeping with the RFC 3411 design decisions to use
        self-contained documents, the RFC that contains the definition
        of this MIB module also includes the elements of procedure
        that are needed for processing the Transport Security Model
        for SNMP.  These MIB objects SHOULD NOT be modified via other
        subsystems or models defined in other documents.  This allows
        the Transport Security Model for SNMP to be designed and
        documented as independent and self-contained, having no direct
        impact on other modules, and this allows this module to be
        upgraded and supplemented as the need arises, and to move
        along the standards track on different time-lines from other
        modules.

        Copyright (c) 2009 IETF Trust and the persons
        identified as authors of the code.  All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, are permitted provided that the
        following conditions are met:

        - Redistributions of source code must retain the above copyright
          notice, this list of conditions and the following disclaimer.

        - Redistributions in binary form must reproduce the above
          copyright notice, this list of conditions and the following
          disclaimer in the documentation and/or other materials
          provided with the distribution.
```

```
        - Neither the name of Internet Society, IETF or IETF Trust,
          nor the names of specific contributors, may be used to endorse
          or promote products derived from this software without
          specific prior written permission.

        THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
        CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES,
        INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
        MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
        DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
        CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
        SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
        NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
        LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
        HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
        CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
        OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
        EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

        This version of this MIB module is part of RFC 5591;
        see the RFC itself for full legal notices."

    REVISION    "200906090000Z"
    DESCRIPTION "The initial version, published in RFC 5591."

    ::= { mib-2 190 }

-- ------------------------------------------------------------ --
-- subtrees in the SNMP-TSM-MIB
-- ------------------------------------------------------------ --

snmpTsmNotifications OBJECT IDENTIFIER ::= { snmpTsmMIB 0 }
snmpTsmMIBObjects    OBJECT IDENTIFIER ::= { snmpTsmMIB 1 }
snmpTsmConformance   OBJECT IDENTIFIER ::= { snmpTsmMIB 2 }

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

-- Statistics for the Transport Security Model

snmpTsmStats         OBJECT IDENTIFIER ::= { snmpTsmMIBObjects 1 }

snmpTsmInvalidCaches OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of incoming messages dropped because the
```

```
                tmStateReference referred to an invalid cache.
               "
    ::= { snmpTsmStats 1 }

snmpTsmInadequateSecurityLevels OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of incoming messages dropped because
                 the securityLevel asserted by the Transport Model was
                 less than the securityLevel requested by the
                 application.
                "
    ::= { snmpTsmStats 2 }

snmpTsmUnknownPrefixes OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of messages dropped because
                 snmpTsmConfigurationUsePrefix was set to true and
                 there is no known prefix for the specified transport
                 domain.
                "
    ::= { snmpTsmStats 3 }

snmpTsmInvalidPrefixes OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of messages dropped because
                 the securityName associated with an outgoing message
                 did not contain a valid transport domain prefix.
                "
    ::= { snmpTsmStats 4 }

-- ------------------------------------------------------------
-- Configuration
-- ------------------------------------------------------------

-- Configuration for the Transport Security Model

snmpTsmConfiguration   OBJECT IDENTIFIER ::= { snmpTsmMIBObjects 2 }

snmpTsmConfigurationUsePrefix OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-write
    STATUS      current
```

```
    DESCRIPTION "If this object is set to true, then securityNames
                 passing to and from the application are expected to
                 contain a transport-domain-specific prefix.  If this
                 object is set to true, then a domain-specific prefix
                 will be added by the TSM to the securityName for
                 incoming messages and removed from the securityName
                 when processing outgoing messages.  Transport domains
                 and prefixes are maintained in a registry by IANA.
                 This object SHOULD persist across system reboots.
                "
    DEFVAL { false }
    ::= { snmpTsmConfiguration 1 }

-- ----------------------------------------------------------------
-- snmpTsmMIB - Conformance Information
-- ----------------------------------------------------------------

snmpTsmCompliances OBJECT IDENTIFIER ::= { snmpTsmConformance 1 }

snmpTsmGroups      OBJECT IDENTIFIER ::= { snmpTsmConformance 2 }

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

snmpTsmCompliance MODULE-COMPLIANCE
    STATUS      current
    DESCRIPTION "The compliance statement for SNMP engines that support
                 the SNMP-TSM-MIB.
                "
    MODULE
        MANDATORY-GROUPS { snmpTsmGroup }
    ::= { snmpTsmCompliances 1 }

-- ----------------------------------------------------------------
-- Units of conformance
-- ----------------------------------------------------------------
snmpTsmGroup OBJECT-GROUP
    OBJECTS {
        snmpTsmInvalidCaches,
        snmpTsmInadequateSecurityLevels,
        snmpTsmUnknownPrefixes,
        snmpTsmInvalidPrefixes,
        snmpTsmConfigurationUsePrefix
    }
    STATUS      current
    DESCRIPTION "A collection of objects for maintaining
                 information of an SNMP engine that implements
```

```
                the SNMP Transport Security Model.
              "

    ::= { snmpTsmGroups 2 }

END
```

## 8. Security Considerations

This document describes a Security Model, compatible with the RFC 3411 architecture, that permits SNMP to utilize security services provided through an SNMP Transport Model. The Transport Security Model relies on Transport Models for mutual authentication, binding of keys, confidentiality, and integrity.

The Transport Security Model relies on secure Transport Models to provide an authenticated principal identifier and an assertion of whether authentication and privacy are used during transport. This Security Model SHOULD always be used with Transport Models that provide adequate security, but "adequate security" is a configuration and/or run-time decision of the operator or management application. The security threats and how these threats are mitigated should be covered in detail in the specifications of the Transport Models and the underlying secure transports.

An authenticated principal identifier (securityName) is used in SNMP applications for purposes such as access control, notification generation, and proxy forwarding. This Security Model supports multiple Transport Models. Operators might judge some transports to be more secure than others, so this Security Model can be configured to prepend a prefix to the securityName to indicate the Transport Model used to authenticate the principal. Operators can use the prefixed securityName when making application decisions about levels of access.

### 8.1. MIB Module Security

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:

- The snmpTsmConfigurationUsePrefix object could be modified, creating a denial of service or authorizing SNMP messages that would not have previously been authorized by an Access Control Model (e.g., the View-based Access Control Model (VACM)).

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:

- All the counters in this module refer to configuration errors and do not expose sensitive information.

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 USM and Transport Security Model 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.

## 9. IANA Considerations

IANA has assigned:

1. An SMI number (190) with a prefix of mib-2 in the MIB module registry for the MIB module in this document.

2. A value (4) to identify the Transport Security Model, in the Security Models registry of the SNMP Number Spaces registry. This results in the following table of values:

| Value | Description | References |
|---|---|---|
| 0 | reserved for 'any' | [RFC3411] |
| 1 | reserved for SNMPv1 | [RFC3411] |
| 2 | reserved for SNMPv2c | [RFC3411] |
| 3 | User-Based Security Model (USM) | [RFC3411] |
| 4 | Transport Security Model (TSM) | [RFC5591] |

## 10. Acknowledgments

The editors would like to thank Jeffrey Hutzelman for sharing his SSH insights and Dave Shield for an outstanding job wordsmithing the existing document to improve organization and clarity.

Additionally, helpful document reviews were received from Juergen Schoenwaelder.

## 11. References

### 11.1. Normative References

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

[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.

[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.

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

[RFC5590] Harrington, D. and J. Schoenwaelder, "Transport Subsystem for the Simple Network Management Protocol (SNMP)", RFC 5590, June 2009.

## 11.2. 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.

[RFC3415] Wijnen, B., Presuhn, R., and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3415, December 2002.

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

[RFC3584] Frye, R., Levi, D., Routhier, S., and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", BCP 74, RFC 3584, August 2003.

[RFC5592] Harrington, D., Salowey, J., and W. Hardaker, "Secure Shell Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5592, June 2009.

## Appendix A. Notification Tables Configuration

The SNMP-TARGET-MIB and SNMP-NOTIFICATION-MIB [RFC3413] are used to configure notification originators with the destinations to which notifications should be sent.

Most of the configuration is Security-Model-independent and Transport-Model-independent.

The values we will use in the examples for the five model-independent security and transport parameters are:

transportDomain = snmpSSHDomain

transportAddress = 192.0.2.1:5162

securityModel = Transport Security Model

securityName = alice

securityLevel = authPriv

The following example will configure the notification originator to send informs to a notification receiver at 192.0.2.1:5162 using the securityName "alice". "alice" is the name for the recipient from the standpoint of the notification originator and is used for processing access controls before sending a notification.

The columns marked with an "*" are the items that are Security-Model-specific or Transport-Model-specific.

The configuration for the "alice" settings in the SNMP-VIEW-BASED-ACM-MIB objects are not shown here for brevity. First, we configure which type of notification will be sent for this taglist (toCRTag). In this example, we choose to send an Inform.

snmpNotifyTable row:

| | |
|---|---|
| snmpNotifyName | CRNotif |
| snmpNotifyTag | toCRTag |
| snmpNotifyType | inform |
| snmpNotifyStorageType | nonVolatile |
| snmpNotifyColumnStatus | createAndGo |

Then we configure a transport address to which notifications associated with this taglist will be sent, and we specify which snmpTargetParamsEntry will be used (toCR) when sending to this transport address.

```
        snmpTargetAddrTable row:
           snmpTargetAddrName              toCRAddr
         *  snmpTargetAddrTDomain           snmpSSHDomain
         *  snmpTargetAddrTAddress          192.0.2.1:5162
           snmpTargetAddrTimeout           1500
           snmpTargetAddrRetryCount        3
           snmpTargetAddrTagList           toCRTag
           snmpTargetAddrParams            toCR   (MUST match below)
           snmpTargetAddrStorageType       nonVolatile
           snmpTargetAddrColumnStatus      createAndGo
```

Then we configure which principal at the host will receive the notifications associated with this taglist. Here, we choose "alice", who uses the Transport Security Model.

```
      snmpTargetParamsTable row:
           snmpTargetParamsName            toCR
           snmpTargetParamsMPModel         SNMPv3
         *  snmpTargetParamsSecurityModel   TransportSecurityModel
           snmpTargetParamsSecurityName    "alice"
           snmpTargetParamsSecurityLevel   authPriv
           snmpTargetParamsStorageType     nonVolatile
           snmpTargetParamsRowStatus       createAndGo
```

## A.1. Transport Security Model Processing for Notifications

The Transport Security Model is called using the generateRequestMsg() ASI, with the following parameters (those with an * are from the above tables):

```
 statusInformation =                -- success or errorIndication
       generateRequestMsg(
       IN   messageProcessingModel  -- *snmpTargetParamsMPModel
       IN   globalData              -- message header, admin data
       IN   maxMessageSize          -- of the sending SNMP entity
       IN   transportDomain         -- *snmpTargetAddrTDomain
       IN   transportAddress        -- *snmpTargetAddrTAddress
       IN   securityModel           -- *snmpTargetParamsSecurityModel
       IN   securityEngineID        -- immaterial; TSM will ignore.
       IN   securityName            -- snmpTargetParamsSecurityName
       IN   securityLevel           -- *snmpTargetParamsSecurityLevel
       IN   scopedPDU               -- message (plaintext) payload
       OUT  securityParameters      -- filled in by Security Module
       OUT  wholeMsg                -- complete generated message
       OUT  wholeMsgLength          -- length of generated message
       OUT  tmStateReference        -- reference to transport info
            )
```

The Transport Security Model will determine the Transport Model based on the snmpTargetAddrTDomain. The selected Transport Model will select the appropriate transport connection using the tmStateReference cache created from the values of snmpTargetAddrTAddress, snmpTargetParamsSecurityName, and snmpTargetParamsSecurityLevel.

## Appendix B. Processing Differences between USM and Secure Transport

USM and secure transports differ in the processing order and responsibilities within the RFC 3411 architecture. While the steps are the same, they occur in a different order and might be done by different subsystems. The following lists illustrate the difference in the flow and the responsibility for different processing steps for incoming messages when using USM and when using a secure transport. (These lists are simplified for illustrative purposes, and do not represent all details of processing. Transport Models MUST provide the detailed elements of procedure.)

With USM, SNMPv1, and SNMPv2c Security Models, security processing starts when the Message Processing Model decodes portions of the ASN.1 message to extract header fields that are used to determine which Security Model will process the message to perform authentication, decryption, timeliness checking, integrity checking, and translation of parameters to model-independent parameters. By comparison, a secure transport performs those security functions on the message, before the ASN.1 is decoded.

Step 6 cannot occur until after decryption occurs. Steps 6 and beyond are the same for USM and a secure transport.

### B.1. USM and the RFC 3411 Architecture

1) Decode the ASN.1 header (Message Processing Model).

2) Determine the SNMP Security Model and parameters (Message Processing Model).

3) Verify securityLevel (Security Model).

4) Translate parameters to model-independent parameters (Security Model).

5) Authenticate the principal, check message integrity and timeliness, and decrypt the message (Security Model).

6) Determine the pduType in the decrypted portions (Message Processing Model).

7) Pass on the decrypted portions with model-independent parameters.

## B.2. Transport Subsystem and the RFC 3411 Architecture

1) Authenticate the principal, check integrity and timeliness of the message, and decrypt the message (Transport Model).

2) Translate parameters to model-independent parameters (Transport Model).

3) Decode the ASN.1 header (Message Processing Model).

4) Determine the SNMP Security Model and parameters (Message Processing Model).

5) Verify securityLevel (Security Model).

6) Determine the pduType in the decrypted portions (Message Processing Model).

7) Pass on the decrypted portions with model-independent security parameters.

If a message is secured using a secure transport layer, then the Transport Model will provide the translation from the authenticated identity (e.g., an SSH user name) to a human-friendly identifier (tmSecurityName) in step 2. The Security Model will provide a mapping from that identifier to a model-independent securityName.

## Authors' Addresses

David Harrington
Huawei Technologies (USA)
1700 Alma Dr. Suite 100
Plano, TX 75075
USA

Phone: +1 603 436 8634
EMail: ietfdbh@comcast.net

Wes Hardaker
Cobham Analytic Solutions
P.O. Box 382
Davis, CA 95617
US

Phone: +1 530 792 1913
EMail: ietf@hardakers.net

Internet Engineering Task Force (IETF)
Request for Comments: 6353
Obsoletes: 5953
Category: Standards Track
ISSN: 2070-1721

W. Hardaker
SPARTA, Inc.

July 2011

# Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)

## Abstract

This document describes a Transport Model for the Simple Network Management Protocol (SNMP), that uses either the Transport Layer Security protocol or the Datagram Transport Layer Security (DTLS) protocol. The TLS and DTLS protocols provide authentication and privacy services for SNMP applications. This document describes how the TLS Transport Model (TLSTM) implements the needed features of an SNMP Transport Subsystem to make this protection possible in an interoperable way.

This Transport Model is designed to meet the security and operational needs of network administrators. It supports the sending of SNMP messages over TLS/TCP and DTLS/UDP. The TLS mode can make use of TCP's improved support for larger packet sizes and the DTLS mode provides potentially superior operation in environments where a connectionless (e.g., UDP) transport is preferred. Both TLS and DTLS integrate well into existing public keying infrastructures.

This document also defines a portion of the Management Information Base (MIB) for use with network management protocols. In particular, it defines objects for managing the TLS Transport Model for SNMP.

## Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6353.

Table of Contents

## 1. Introduction

It is important to understand the modular SNMPv3 architecture as defined by [RFC3411] and enhanced by the Transport Subsystem [RFC5590]. It is also important to understand the terminology of the SNMPv3 architecture in order to understand where the Transport Model described in this document fits into the architecture and how it interacts with the other architecture subsystems. For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to Section 7 of [RFC3410].

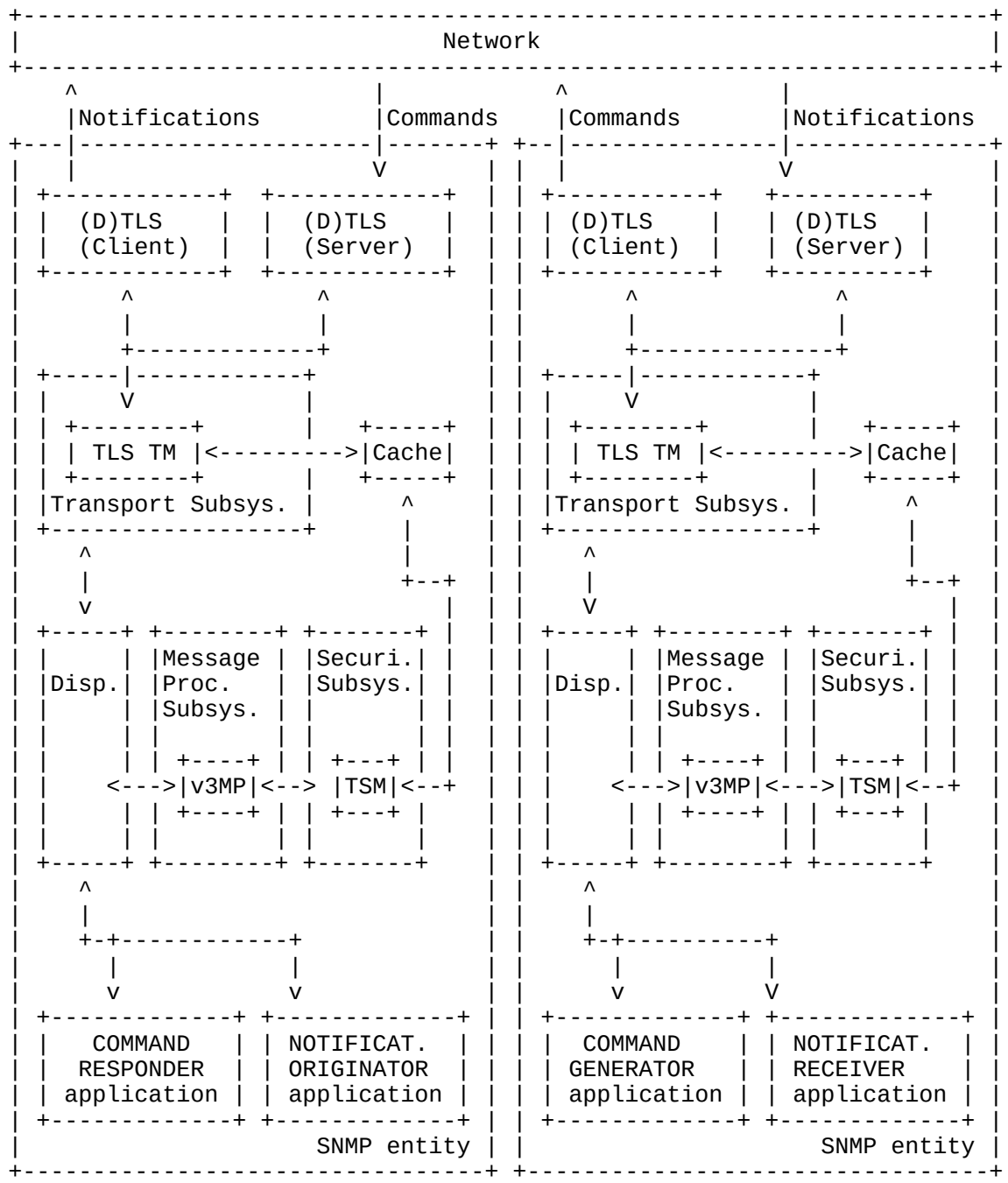
This document describes a Transport Model that makes use of the Transport Layer Security (TLS) [RFC5246] and the Datagram Transport Layer Security (DTLS) Protocol [RFC4347], within a Transport Subsystem [RFC5590]. DTLS is the datagram variant of the Transport Layer Security (TLS) protocol [RFC5246]. The Transport Model in this document is referred to as the Transport Layer Security Transport Model (TLSTM). TLS and DTLS take advantage of the X.509 public keying infrastructure [RFC5280]. While (D)TLS supports multiple authentication mechanisms, this document only discusses X.509 certificate-based authentication. Although other forms of authentication are possible, they are outside the scope of this specification. This transport model is designed to meet the security and operational needs of network administrators, operating in both environments where a connectionless (e.g., UDP) transport is preferred and in environments where large quantities of data need to be sent (e.g., over a TCP-based stream). Both TLS and DTLS integrate well into existing public keying infrastructures. This document supports sending of SNMP messages over TLS/TCP and DTLS/UDP.

This document also defines a portion of the Management Information Base (MIB) for use with network management protocols. In particular, it defines objects for managing the TLS Transport Model for SNMP.

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: [RFC2578], [RFC2579], and [RFC2580].

The diagram shown below gives a conceptual overview of two SNMP entities communicating using the TLS Transport Model (shown as "TLSTM"). One entity contains a command responder and notification originator application, and the other a command generator and notification receiver application. It should be understood that this particular mix of application types is an example only and other combinations are equally valid.

Note: this diagram shows the Transport Security Model (TSM) being used as the security model that is defined in [RFC5591].


Network
Notifications
Commands
Commands
Notifications
(D)TLS (Client)
(D)TLS (Server)
(D)TLS (Client)
(D)TLS (Server)
TLS TM
Cache
Transport Subsys.
Disp.
Message Proc. Subsys.
Securi. Subsys.
v3MP
TSM
COMMAND RESPONDER application
NOTIFICAT. ORIGINATOR application
COMMAND GENERATOR application
NOTIFICAT. RECEIVER application
SNMP entity
SNMP entity

### 1.1. Conventions

For consistency with SNMP-related specifications, this document favors terminology as defined in STD 62, rather than favoring terminology that is consistent with non-SNMP specifications. This is consistent with the IESG decision to not require the SNMPv3 terminology be modified to match the usage of other non-SNMP specifications when SNMPv3 was advanced to a Full Standard.

"Authentication" in this document typically refers to the English meaning of "serving to prove the authenticity of" the message, not data source authentication or peer identity authentication.

The terms "manager" and "agent" are not used in this document because, in the [RFC3411] architecture, all SNMP entities have the capability of acting as manager, agent, or both depending on the SNMP application types supported in the implementation. Where distinction is required, the application names of command generator, command responder, notification originator, notification receiver, and proxy forwarder are used. See "SNMP Applications" [RFC3413] for further information.

Large portions of this document simultaneously refer to both TLS and DTLS when discussing TLSTM components that function equally with either protocol. "(D)TLS" is used in these places to indicate that the statement applies to either or both protocols as appropriate. When a distinction between the protocols is needed, they are referred to independently through the use of "TLS" or "DTLS". The Transport Model, however, is named "TLS Transport Model" and refers not to the TLS or DTLS protocol but to the specification in this document, which includes support for both TLS and DTLS.

Throughout this document, the terms "client" and "server" are used to refer to the two ends of the (D)TLS transport connection. The client actively opens the (D)TLS connection, and the server passively listens for the incoming (D)TLS connection. An SNMP entity may act as a (D)TLS client or server or both, depending on the SNMP applications supported.

The User-Based Security Model (USM) [RFC3414] is a mandatory-to-implement Security Model in STD 62. While (D)TLS and USM frequently refer to a user, the terminology preferred in RFC 3411 and in this memo is "principal". A 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, or a combination of these within an administrative domain.

Throughout this document, the term "session" is used to refer to a secure association between two TLS Transport Models that permits the transmission of one or more SNMP messages within the lifetime of the session. The (D)TLS protocols also have an internal notion of a session and although these two concepts of a session are related, when the term "session" is used this document is referring to the TLSTM's specific session and not directly to the (D)TLS protocol's session.

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

### 1.2. Changes Since RFC 5953

This document obsoletes [RFC5953].

Since the publication of RFC 5953, a few editorial errata have been noted. These errata are posted on the RFC Editor web site. These errors have been corrected in this document.

This document updates the references to RFC 3490 (IDNA 2003) to [RFC5890] (IDNA 2008), because RFC 3490 was obsoleted by RFC 5890.

References to RFC 1033 were replaced with references to [RFC1123].

Added informative reference to 5953.

Updated MIB dates and revision date.

## 2. The Transport Layer Security Protocol

(D)TLS provides authentication, data message integrity, and privacy at the transport layer (see [RFC4347]).

The primary goals of the TLS Transport Model are to provide privacy, peer identity authentication, and data integrity between two communicating SNMP entities. The TLS and DTLS protocols provide a secure transport upon which the TLSTM is based. Please refer to [RFC5246] and [RFC4347] for complete descriptions of the protocols.

## 3. How the TLSTM Fits into the Transport Subsystem

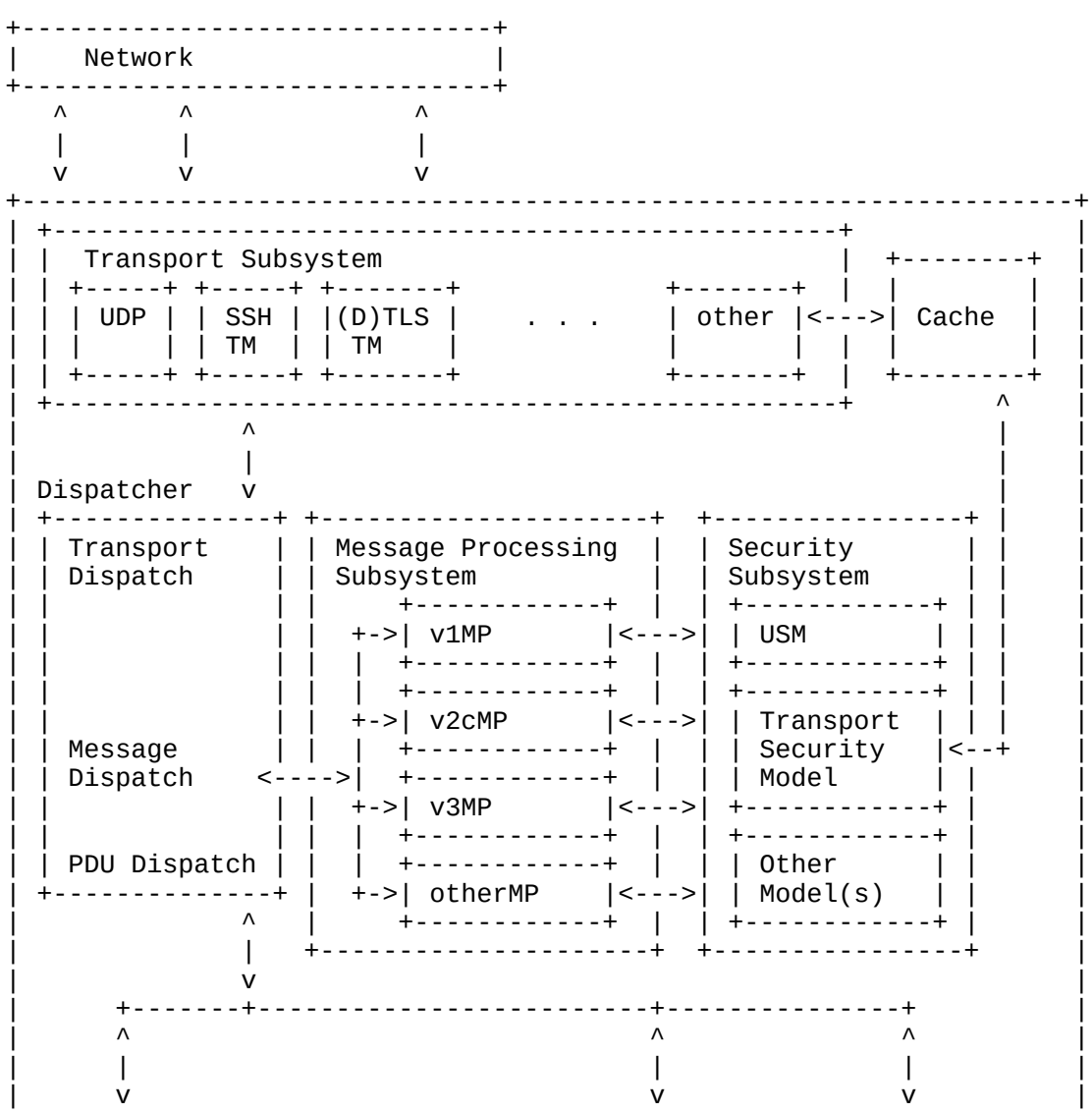
A transport model is a component of the Transport Subsystem. The TLS Transport Model thus fits between the underlying (D)TLS transport layer and the Message Dispatcher [RFC3411] component of the SNMP engine.

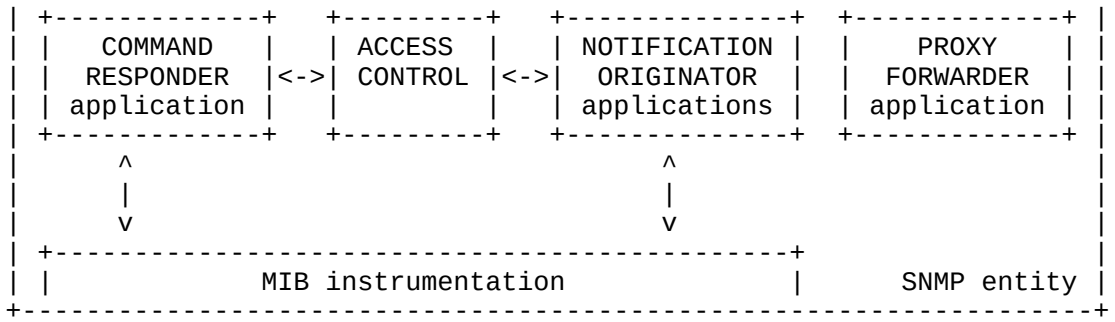
The TLS Transport Model will establish a session between itself and the TLS Transport Model of another SNMP engine. The sending transport model passes unencrypted and unauthenticated messages from the Dispatcher to (D)TLS to be encrypted and authenticated, and the receiving transport model accepts decrypted and authenticated/ integrity-checked incoming messages from (D)TLS and passes them to the Dispatcher.

After a TLS Transport Model session is established, SNMP messages can conceptually be sent through the session from one SNMP message Dispatcher to another SNMP Message Dispatcher. If multiple SNMP messages are needed to be passed between two SNMP applications they MAY be passed through the same session. A TLSTM implementation engine MAY choose to close the session to conserve resources.

The TLS Transport Model of an SNMP engine will perform the translation between (D)TLS-specific security parameters and SNMP-specific, model-independent parameters.

The diagram below depicts where the TLS Transport Model (shown as "(D)TLS TM") fits into the architecture described in RFC 3411 and the Transport Subsystem:






### 3.1. Security Capabilities of This Model

#### 3.1.1. Threats

The TLS Transport Model provides protection against the threats identified by the RFC 3411 architecture [RFC3411]:

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

   (D)TLS provides verification that the content of each received message has not been modified during its transmission through the network, data has not been altered or destroyed in an unauthorized manner, and data sequences have not been altered to an extent greater than can occur non-maliciously.

2. Masquerade - The masquerade threat is the danger that management operations unauthorized for a given principal may be attempted by assuming the identity of another principal that has the appropriate authorizations.

   The TLSTM verifies the identity of the (D)TLS server through the use of the (D)TLS protocol and X.509 certificates. A TLS Transport Model implementation MUST support the authentication of both the server and the client.

3. Message stream modification - The re-ordering, delay, or replay of messages can and does occur through the natural operation of many connectionless transport services. The message stream modification threat is the danger that messages may be maliciously re-ordered, delayed, or replayed to an extent that is greater than can occur through the natural operation of

connectionless transport services, in order to effect unauthorized management operations.

(D)TLS provides replay protection with a Message Authentication Code (MAC) that includes a sequence number. Since UDP provides no sequencing ability, DTLS uses a sliding window protocol with the sequence number used for replay protection (see [RFC4347]).

4. Disclosure - The disclosure threat is the danger of eavesdropping on the exchanges between SNMP engines.

   (D)TLS provides protection against the disclosure of information to unauthorized recipients or eavesdroppers by allowing for encryption of all traffic between SNMP engines. A TLS Transport Model implementation MUST support message encryption to protect sensitive data from eavesdropping attacks.

5. Denial of Service - The RFC 3411 architecture [RFC3411] states that denial-of-service (DoS) attacks need not be addressed by an SNMP security protocol. However, connectionless transports (like DTLS over UDP) are susceptible to a variety of DoS attacks because they are more vulnerable to spoofed IP addresses. See Section 4.2 for details on how the cookie mechanism is used. Note, however, that this mechanism does not provide any defense against DoS attacks mounted from valid IP addresses.

See Section 9 for more detail on the security considerations associated with the TLSTM and these security threats.

### 3.1.2. Message Protection

The RFC 3411 architecture recognizes three levels of security:

- without authentication and without privacy (noAuthNoPriv)
- with authentication but without privacy (authNoPriv)
- with authentication and with privacy (authPriv)

The TLS Transport Model determines from (D)TLS the identity of the authenticated principal, the transport type, and the transport address associated with an incoming message. The TLS Transport Model provides the identity and destination type and address to (D)TLS for outgoing messages.

When an application requests a session for a message, it also requests a security level for that session. The TLS Transport Model MUST ensure that the (D)TLS connection provides security at least as

high as the requested level of security. How the security level is translated into the algorithms used to provide data integrity and privacy is implementation dependent. However, the NULL integrity and encryption algorithms MUST NOT be used to fulfill security level requests for authentication or privacy. Implementations MAY choose to force (D)TLS to only allow cipher_suites that provide both authentication and privacy to guarantee this assertion.

If a suitable interface between the TLS Transport Model and the (D)TLS Handshake Protocol is implemented to allow the selection of security-level-dependent algorithms (for example, a security level to cipher_suites mapping table), then different security levels may be utilized by the application.

The authentication, integrity, and privacy algorithms used by the (D)TLS Protocols may vary over time as the science of cryptography continues to evolve and the development of (D)TLS continues over time. Implementers are encouraged to plan for changes in operator trust of particular algorithms. Implementations SHOULD offer configuration settings for mapping algorithms to SNMPv3 security levels.

#### 3.1.3. (D)TLS Connections

(D)TLS connections are opened by the TLS Transport Model during the elements of procedure for an outgoing SNMP message. Since the sender of a message initiates the creation of a (D)TLS connection if needed, the (D)TLS connection will already exist for an incoming message.

Implementations MAY choose to instantiate (D)TLS connections in anticipation of outgoing messages. This approach might be useful to ensure that a (D)TLS connection to a given target can be established before it becomes important to send a message over the (D)TLS connection. Of course, there is no guarantee that a pre-established session will still be valid when needed.

DTLS connections, when used over UDP, are uniquely identified within the TLS Transport Model by the combination of transportDomain, transportAddress, tmSecurityName, and requestedSecurityLevel associated with each session. Each unique combination of these parameters MUST have a locally chosen unique tlstmSessionID for each active session. For further information, see Section 5. TLS over TCP sessions, on the other hand, do not require a unique pairing of address and port attributes since their lower-layer protocols (TCP) already provide adequate session framing. But they must still provide a unique tlstmSessionID for referencing the session.

The tlstmSessionID MUST NOT change during the entire duration of the session from the TLSTM's perspective, and MUST uniquely identify a single session. As an implementation hint: note that the (D)TLS internal SessionID does not meet these requirements, since it can change over the life of the connection as seen by the TLSTM (for example, during renegotiation), and does not necessarily uniquely identify a TLSTM session (there can be multiple TLSTM sessions sharing the same D(TLS) internal SessionID).

## 3.2. Security Parameter Passing

For the (D)TLS server-side, (D)TLS-specific security parameters (i.e., cipher_suites, X.509 certificate fields, IP addresses, and ports) are translated by the TLS Transport Model into security parameters for the TLS Transport Model and security model (e.g., tmSecurityLevel, tmSecurityName, transportDomain, transportAddress). The transport-related and (D)TLS-security-related information, including the authenticated identity, are stored in a cache referenced by tmStateReference.

For the (D)TLS client side, the TLS Transport Model takes input provided by the Dispatcher in the sendMessage() Abstract Service Interface (ASI) and input from the tmStateReference cache. The (D)TLS Transport Model converts that information into suitable security parameters for (D)TLS and establishes sessions as needed.

The elements of procedure in Section 5 discuss these concepts in much greater detail.

## 3.3. Notifications and Proxy

(D)TLS connections may be initiated by (D)TLS clients on behalf of SNMP applications that initiate communications, such as command generators, notification originators, proxy forwarders. Command generators are frequently operated by a human, but notification originators and proxy forwarders are usually unmanned automated processes. The targets to whom notifications and proxied requests should be sent are typically determined and configured by a network administrator.

The SNMP-TARGET-MIB module [RFC3413] contains objects for defining management targets, including transportDomain, transportAddress, securityName, securityModel, and securityLevel parameters, for notification originator, proxy forwarder, and SNMP-controllable command generator applications. Transport domains and transport addresses are configured in the snmpTargetAddrTable, and the securityModel, securityName, and securityLevel parameters are configured in the snmpTargetParamsTable. This document defines a MIB

module that extends the SNMP-TARGET-MIB's snmpTargetParamsTable to specify a (D)TLS client-side certificate to use for the connection.

When configuring a (D)TLS target, the snmpTargetAddrTDomain and snmpTargetAddrTAddress parameters in snmpTargetAddrTable SHOULD be set to the snmpTLSTCPDomain or snmpDTLSUDPDomain object and an appropriate snmpTLSAddress value. When used with the SNMPv3 message processing model, the snmpTargetParamsMPModel column of the snmpTargetParamsTable SHOULD be set to a value of 3. The snmpTargetParamsSecurityName SHOULD be set to an appropriate securityName value, and the snmpTlstmParamsClientFingerprint parameter of the snmpTlstmParamsTable SHOULD be set to a value that refers to a locally held certificate (and the corresponding private key) to be used. Other parameters, for example, cryptographic configuration such as which cipher_suites to use, must come from configuration mechanisms not defined in this document.

The securityName defined in the snmpTargetParamsSecurityName column will be used by the access control model to authorize any notifications that need to be sent.

## 4. Elements of the Model

This section contains definitions required to realize the (D)TLS Transport Model defined by this document.

### 4.1. X.509 Certificates

(D)TLS can make use of X.509 certificates for authentication of both sides of the transport. This section discusses the use of X.509 certificates in the TLSTM.

While (D)TLS supports multiple authentication mechanisms, this document only discusses X.509-certificate-based authentication; other forms of authentication are outside the scope of this specification. TLSTM implementations are REQUIRED to support X.509 certificates.

#### 4.1.1. Provisioning for the Certificate

Authentication using (D)TLS will require that SNMP entities have certificates, either signed by trusted Certification Authorities (CAs), or self signed. Furthermore, SNMP entities will most commonly need to be provisioned with root certificates that represent the list of trusted CAs that an SNMP entity can use for certificate verification. SNMP entities SHOULD also be provisioned with an X.509 certificate revocation mechanism which can be used to verify that a certificate has not been revoked. Trusted public keys from either CA certificates and/or self-signed certificates MUST be installed into

the server through a trusted out-of-band mechanism and their authenticity MUST be verified before access is granted.

Having received a certificate from a connecting TLSTM client, the authenticated tmSecurityName of the principal is derived using the snmpTlstmCertToTSNTable. This table allows mapping of incoming connections to tmSecurityNames through defined transformations. The transformations defined in the SNMP-TLS-TM-MIB include:

- Mapping a certificate's subjectAltName or CommonName components to a tmSecurityName, or
- Mapping a certificate's fingerprint value to a directly specified tmSecurityName

As an implementation hint: implementations may choose to discard any connections for which no potential snmpTlstmCertToTSNTable mapping exists before performing certificate verification to avoid expending computational resources associated with certificate verification.

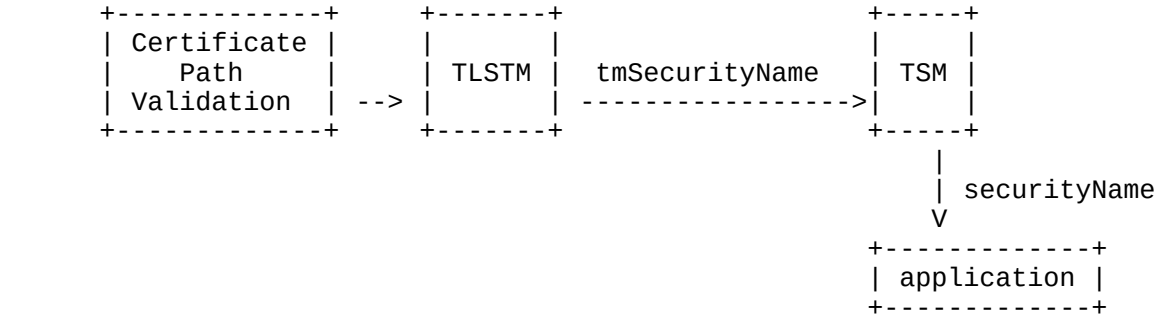
Deployments SHOULD map the "subjectAltName" component of X.509 certificates to the TLSTM specific tmSecurityNames. The authenticated identity can be obtained by the TLS Transport Model by extracting the subjectAltName(s) from the peer's certificate. The receiving application will then have an appropriate tmSecurityName for use by other SNMPv3 components like an access control model.

An example of this type of mapping setup can be found in Appendix A.

This tmSecurityName may be later translated from a TLSTM specific tmSecurityName to an SNMP engine securityName by the security model. A security model, like the TSM security model [RFC5591], may perform an identity mapping or a more complex mapping to derive the securityName from the tmSecurityName offered by the TLS Transport Model.

The standard View-Based Access Control Model (VACM) access control model constrains securityNames to be 32 octets or less in length. A TLSTM generated tmSecurityName, possibly in combination with a messaging or security model that increases the length of the securityName, might cause the securityName length to exceed 32 octets. For example, a 32-octet tmSecurityName derived from an IPv6 address, paired with a TSM prefix, will generate a 36-octet securityName. Such a securityName will not be able to be used with standard VACM or TARGET MIB modules. Operators should be careful to select algorithms and subjectAltNames to avoid this situation.

A pictorial view of the complete transformation process (using the TSM security model for the example) is shown below:




## 4.2. (D)TLS Usage

(D)TLS MUST negotiate a cipher_suite that uses X.509 certificates for authentication, and MUST authenticate both the client and the server. The mandatory-to-implement cipher_suite is specified in the TLS specification [RFC5246].

TLSTM verifies the certificates when the connection is opened (see Section 5.3). For this reason, TLS renegotiation with different certificates MUST NOT be done. That is, implementations MUST either disable renegotiation completely (RECOMMENDED), or they MUST present the same certificate during renegotiation (and MUST verify that the other end presented the same certificate).

For DTLS over UDP, each SNMP message MUST be placed in a single UDP datagram; it MAY be split to multiple DTLS records. In other words, if a single datagram contains multiple DTLS application_data records, they are concatenated when received. The TLSTM implementation SHOULD return an error if the SNMP message does not fit in the UDP datagram, and thus cannot be sent.

For DTLS over UDP, the DTLS server implementation MUST support DTLS cookies ([RFC4347] already requires that clients support DTLS cookies). Implementations are not required to perform the cookie exchange for every DTLS handshake; however, enabling it by default is RECOMMENDED.

For DTLS, replay protection MUST be used.

### 4.3. SNMP Services

This section describes the services provided by the TLS Transport Model with their inputs and outputs. The services are between the Transport Model and the Dispatcher.

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

#### 4.3.1. SNMP Services for an Outgoing Message

The Dispatcher passes the information to the TLS Transport Model using the ASI defined in the Transport Subsystem:

```
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             -- reference to transport state
 )
```

The abstract data elements returned from or passed as parameters into the abstract service primitives are as follows:

statusInformation: An indication of whether the sending of the message was successful. If not, it is an indication of the problem.

destTransportDomain: The transport domain for the associated destTransportAddress. The Transport Model uses this parameter to determine the transport type of the associated destTransportAddress. This document specifies the snmpTLSTCPDomain and the snmpDTLSUDPDomain transport domains.

destTransportAddress: The transport address of the destination TLS Transport Model in a format specified by the SnmpTLSAddress TEXTUAL-CONVENTION.

outgoingMessage: The outgoing message to send to (D)TLS for encapsulation and transmission.

outgoingMessageLength: The length of the outgoingMessage.

tmStateReference: A reference used to pass model-specific and mechanism-specific parameters between the Transport Subsystem and transport-aware Security Models.

#### 4.3.2. SNMP Services for an Incoming Message

The TLS Transport Model processes the received message from the network using the (D)TLS service and then passes it to the Dispatcher using the following ASI:

```
statusInformation =
receiveMessage(
IN   transportDomain              -- origin transport domain
IN   transportAddress             -- origin transport address
IN   incomingMessage              -- the message received
IN   incomingMessageLength        -- its length
IN   tmStateReference             -- reference to transport state
 )
```

The abstract data elements returned from or passed as parameters into the abstract service primitives are as follows:

statusInformation: An indication of whether the passing of the message was successful. If not, it is an indication of the problem.

transportDomain: The transport domain for the associated transportAddress. This document specifies the snmpTLSTCPDomain and the snmpDTLSUDPDomain transport domains.

transportAddress: The transport address of the source of the received message in a format specified by the SnmpTLSAddress TEXTUAL-CONVENTION.

incomingMessage: The whole SNMP message after being processed by (D)TLS.

incomingMessageLength: The length of the incomingMessage.

tmStateReference: A reference used to pass model-specific and mechanism-specific parameters between the Transport Subsystem and transport-aware Security Models.

### 4.4. Cached Information and References

When performing SNMP processing, there are two levels of state information that may need to be retained: the immediate state linking a request-response pair, and potentially longer-term state relating to transport and security. "Transport Subsystem for the Simple Network Management Protocol (SNMP)" [RFC5590] defines general requirements for caches and references.

#### 4.4.1. TLS Transport Model Cached Information

The TLS Transport Model has specific responsibilities regarding the cached information. See the Elements of Procedure in Section 5 for detailed processing instructions on the use of the tmStateReference fields by the TLS Transport Model.

##### 4.4.1.1. tmSecurityName

The tmSecurityName MUST be a human-readable name (in snmpAdminString format) representing the identity that has been set according to the procedures in Section 5. The tmSecurityName MUST be constant for all traffic passing through a single TLSTM session. Messages MUST NOT be sent through an existing (D)TLS connection that was established using a different tmSecurityName.

On the (D)TLS server side of a connection, the tmSecurityName is derived using the procedures described in Section 5.3.2 and the SNMP-TLS-TM-MIB's snmpTlstmCertToTSNTable DESCRIPTION clause.

On the (D)TLS client side of a connection, the tmSecurityName is presented to the TLS Transport Model by the security model through the tmStateReference. This tmSecurityName is typically a copy of or is derived from the securityName that was passed by application (possibly because of configuration specified in the SNMP-TARGET-MIB). The Security Model likely derived the tmSecurityName from the securityName presented to the Security Model by the application (possibly because of configuration specified in the SNMP-TARGET-MIB).

Transport-Model-aware security models derive tmSecurityName from a securityName, possibly configured in MIB modules for notifications and access controls. Transport Models SHOULD use predictable tmSecurityNames so operators will know what to use when configuring MIB modules that use securityNames derived from tmSecurityNames. The TLSTM generates predictable tmSecurityNames based on the configuration found in the SNMP-TLS-TM-MIB's snmpTlstmCertToTSNTable and relies on the network operators to have configured this table appropriately.

#### 4.4.1.2. tmSessionID

The tmSessionID MUST be recorded per message at the time of receipt. When tmSameSecurity is set, the recorded tmSessionID can be used to determine whether the (D)TLS connection available for sending a corresponding outgoing message is the same (D)TLS connection as was used when receiving the incoming message (e.g., a response to a request).

#### 4.4.1.3. Session State

The per-session state that is referenced by tmStateReference may be saved across multiple messages in a Local Configuration Datastore. Additional session/connection state information might also be stored in a Local Configuration Datastore.

## 5. Elements of Procedure

Abstract service interfaces have been defined by [RFC3411] and further augmented by [RFC5590] to describe the conceptual data flows between the various subsystems within an SNMP entity. The TLSTM uses some of these conceptual data flows when communicating between subsystems.

To simplify the elements of procedure, the release of state information is not always explicitly specified. As a general rule, if state information is available when a message gets discarded, the message-state information should also be released. If state information is available when a session is closed, the session state information should also be released. Sensitive information, like cryptographic keys, should be overwritten appropriately prior to being released.

An error indication in statusInformation will typically include the Object Identifier (OID) and value for an incremented error counter. This may be accompanied by the requested securityLevel and the tmStateReference. Per-message context information is not accessible to Transport Models, so for the returned counter OID and value, contextEngine would be set to the local value of snmpEngineID and contextName to the default context for error counters.

### 5.1. Procedures for an Incoming Message

This section describes the procedures followed by the (D)TLS Transport Model when it receives a (D)TLS protected packet. The required functionality is broken into two different sections.

Section 5.1.1 describes the processing required for de-multiplexing multiple DTLS connections, which is specifically needed for DTLS over UDP sessions. It is assumed that TLS protocol implementations already provide appropriate message demultiplexing.

Section 5.1.2 describes the transport processing required once the (D)TLS processing has been completed. This will be needed for all (D)TLS-based connections.

### 5.1.1. DTLS over UDP Processing for Incoming Messages

Demultiplexing of incoming packets into separate DTLS sessions MUST be implemented. For connection-oriented transport protocols, such as TCP, the transport protocol takes care of demultiplexing incoming packets to the right connection. For DTLS over UDP, this demultiplexing will either need to be done within the DTLS implementation, if supported, or by the TLSTM implementation.

Like TCP, DTLS over UDP uses the four-tuple <source IP, destination IP, source port, destination port> for identifying the connection (and relevant DTLS connection state). This means that when establishing a new session, implementations MUST use a different UDP source port number for each active connection to a remote destination IP-address/port-number combination to ensure the remote entity can disambiguate between multiple connections.

If demultiplexing received UDP datagrams to DTLS connection state is done by the TLSTM implementation (instead of the DTLS implementation), the steps below describe one possible method to accomplish this.

The important output results from the steps in this process are the remote transport address, incomingMessage, incomingMessageLength, and the tlstmSessionID.

1) The TLS Transport Model examines the raw UDP message, in an implementation-dependent manner.

2) The TLS Transport Model queries the Local Configuration Datastore (LCD) (see [RFC3411], Section 3.4.2) using the transport parameters (source and destination IP addresses and ports) to determine if a session already exists.

   2a) If a matching entry in the LCD does not exist, then the UDP packet is passed to the DTLS implementation for processing. If the DTLS implementation decides to continue with the connection and allocate state for it, it returns a new DTLS connection handle (an implementation dependent detail). In

this case, TLSTM selects a new tlstmSessionId, and caches this and the DTLS connection handle as a new entry in the LCD (indexed by the transport parameters). If the DTLS implementation returns an error or does not allocate connection state (which can happen with the stateless cookie exchange), processing stops.

2b) If a session does exist in the LCD, then its DTLS connection handle (an implementation dependent detail) and its tlstmSessionId is extracted from the LCD. The UDP packet and the connection handle are passed to the DTLS implementation. If the DTLS implementation returns success but does not return an incomingMessage and an incomingMessageLength, then processing stops (this is the case when the UDP datagram contained DTLS handshake messages, for example). If the DTLS implementation returns an error, then processing stops.

3) Retrieve the incomingMessage and an incomingMessageLength from DTLS. These results and the tlstmSessionID are used below in Section 5.1.2 to complete the processing of the incoming message.

### 5.1.2. Transport Processing for Incoming SNMP Messages

The procedures in this section describe how the TLS Transport Model should process messages that have already been properly extracted from the (D)TLS stream. Note that care must be taken when processing messages originating from either TLS or DTLS to ensure they're complete and single. For example, multiple SNMP messages can be passed through a single DTLS message and partial SNMP messages may be received from a TLS stream. These steps describe the processing of a singular SNMP message after it has been delivered from the (D)TLS stream.

1) Determine the tlstmSessionID for the incoming message. The tlstmSessionID MUST be a unique session identifier for this (D)TLS connection. The contents and format of this identifier are implementation dependent as long as it is unique to the session. A session identifier MUST NOT be reused until all references to it are no longer in use. The tmSessionID is equal to the tlstmSessionID discussed in Section 5.1.1. tmSessionID refers to the session identifier when stored in the tmStateReference and tlstmSessionID refers to the session identifier when stored in the LCD. They MUST always be equal when processing a given session's traffic.

If this is the first message received through this session, and the session does not have an assigned tlstmSessionID yet, then the snmpTlstmSessionAccepts counter is incremented and a tlstmSessionID for the session is created. This will only happen on the server side of a connection because a client would have already assigned a tlstmSessionID during the openSession() invocation. Implementations may have performed the procedures described in Section 5.3.2 prior to this point or they may perform them now, but the procedures described in Section 5.3.2 MUST be performed before continuing beyond this point.

2) Create a tmStateReference cache for the subsequent reference and assign the following values within it:

   tmTransportDomain = snmpTLSTCPDomain or snmpDTLSUDPDomain as appropriate.

   tmTransportAddress = The address from which the message originated.

   tmSecurityLevel = The derived tmSecurityLevel for the session, as discussed in Sections 3.1.2 and 5.3.

   tmSecurityName = The derived tmSecurityName for the session as discussed in Section 5.3. This value MUST be constant during the lifetime of the session.

   tmSessionID = The tlstmSessionID described in step 1 above.

3) The incomingMessage and incomingMessageLength are assigned values from the (D)TLS processing.

4) The TLS Transport Model passes the transportDomain, transportAddress, incomingMessage, and incomingMessageLength to the Dispatcher using the receiveMessage ASI:

```
statusInformation =
receiveMessage(
IN   transportDomain     -- snmpTLSTCPDomain or snmpDTLSUDPDomain,
IN   transportAddress    -- address for the received message
IN   incomingMessage        -- the whole SNMP message from (D)TLS
IN   incomingMessageLength  -- the length of the SNMP message
IN   tmStateReference    -- transport info
 )
```

## 5.2. Procedures for an Outgoing SNMP Message

The Dispatcher sends a message to the TLS Transport Model using the following ASI:

```
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              -- transport info
)
```

This section describes the procedure followed by the TLS Transport Model whenever it is requested through this ASI to send a message.

1) If tmStateReference does not refer to a cache containing values for tmTransportDomain, tmTransportAddress, tmSecurityName, tmRequestedSecurityLevel, and tmSameSecurity, then increment the snmpTlstmSessionInvalidCaches counter, discard the message, and return the error indication in the statusInformation. Processing of this message stops.

2) Extract the tmSessionID, tmTransportDomain, tmTransportAddress, tmSecurityName, tmRequestedSecurityLevel, and tmSameSecurity values from the tmStateReference. Note: the tmSessionID value may be undefined if no session exists yet over which the message can be sent.

3) If tmSameSecurity is true and tmSessionID is either undefined or refers to a session that is no longer open, then increment the snmpTlstmSessionNoSessions counter, discard the message, and return the error indication in the statusInformation. Processing of this message stops.

4) If tmSameSecurity is false and tmSessionID refers to a session that is no longer available, then an implementation SHOULD open a new session, using the openSession() ASI (described in greater detail in step 5b). Instead of opening a new session an implementation MAY return an snmpTlstmSessionNoSessions error to the calling module and stop the processing of the message.

5) If tmSessionID is undefined, then use tmTransportDomain, tmTransportAddress, tmSecurityName, and tmRequestedSecurityLevel to see if there is a corresponding entry in the LCD suitable to send the message over.

5a) If there is a corresponding LCD entry, then this session will be used to send the message.

5b) If there is no corresponding LCD entry, then open a session using the openSession() ASI (discussed further in Section 5.3.1). Implementations MAY wish to offer message buffering to prevent redundant openSession() calls for the same cache entry. If an error is returned from openSession(), then discard the message, discard the tmStateReference, increment the snmpTlstmSessionOpenErrors, return an error indication to the calling module, and stop the processing of the message.

6) Using either the session indicated by the tmSessionID (if there was one) or the session resulting from a previous step (4 or 5), pass the outgoingMessage to (D)TLS for encapsulation and transmission.

## 5.3. Establishing or Accepting a Session

Establishing a (D)TLS connection as either a client or a server requires slightly different processing. The following two sections describe the necessary processing steps.

### 5.3.1. Establishing a Session as a Client

The TLS Transport Model provides the following primitive for use by a client to establish a new (D)TLS connection:

```
statusInformation =          -- errorIndication or success
openSession(
IN   tmStateReference        -- transport information to be used
OUT  tmStateReference        -- transport information to be used
IN   maxMessageSize          -- of the sending SNMP entity
)
```

The following describes the procedure to follow when establishing an SNMP over a (D)TLS connection between SNMP engines for exchanging SNMP messages. This process is followed by any SNMP client's engine when establishing a session for subsequent use.

This procedure MAY be done automatically for an SNMP application that initiates a transaction, such as a command generator, a notification originator, or a proxy forwarder.

1) The snmpTlstmSessionOpens counter is incremented.

2) The client selects the appropriate certificate and cipher_suites for the key agreement based on the tmSecurityName and the tmRequestedSecurityLevel for the session. For sessions being established as a result of an SNMP-TARGET-MIB based operation, the certificate will potentially have been identified via the snmpTlstmParamsTable mapping and the cipher_suites will have to be taken from a system-wide or implementation-specific configuration. If no row in the snmpTlstmParamsTable exists, then implementations MAY choose to establish the connection using a default client certificate available to the application. Otherwise, the certificate and appropriate cipher_suites will need to be passed to the openSession() ASI as supplemental information or configured through an implementation-dependent mechanism. It is also implementation-dependent and possibly policy-dependent how tmRequestedSecurityLevel will be used to influence the security capabilities provided by the (D)TLS connection. However this is done, the security capabilities provided by (D)TLS MUST be at least as high as the level of security indicated by the tmRequestedSecurityLevel parameter. The actual security level of the session is reported in the tmStateReference cache as tmSecurityLevel. For (D)TLS to provide strong authentication, each principal acting as a command generator SHOULD have its own certificate.

3) Using the destTransportDomain and destTransportAddress values, the client will initiate the (D)TLS handshake protocol to establish session keys for message integrity and encryption.

   If the attempt to establish a session is unsuccessful, then snmpTlstmSessionOpenErrors is incremented, an error indication is returned, and processing stops. If the session failed to open because the presented server certificate was unknown or invalid, then the snmpTlstmSessionUnknownServerCertificate or snmpTlstmSessionInvalidServerCertificates MUST be incremented and an snmpTlstmServerCertificateUnknown or snmpTlstmServerInvalidCertificate notification SHOULD be sent as appropriate. Reasons for server certificate invalidation include, but are not limited to, cryptographic validation failures and an unexpected presented certificate identity.

4) The (D)TLS client MUST then verify that the (D)TLS server’s presented certificate is the expected certificate. The (D)TLS client MUST NOT transmit SNMP messages until the server certificate has been authenticated, the client certificate has been transmitted, and the TLS connection has been fully established.

If the connection is being established from a configuration based on SNMP-TARGET-MIB configuration, then the snmpTlstmAddrTable DESCRIPTION clause describes how the verification is done (using either a certificate fingerprint, or an identity authenticated via certification path validation).

If the connection is being established for reasons other than configuration found in the SNMP-TARGET-MIB, then configuration and procedures outside the scope of this document should be followed. Configuration mechanisms SHOULD be similar in nature to those defined in the snmpTlstmAddrTable to ensure consistency across management configuration systems. For example, a command-line tool for generating SNMP GETs might support specifying either the server's certificate fingerprint or the expected host name as a command-line argument.

5) (D)TLS provides assurance that the authenticated identity has been signed by a trusted configured Certification Authority. If verification of the server's certificate fails in any way (for example, because of failures in cryptographic verification or the presented identity did not match the expected named entity), then the session establishment MUST fail, and the snmpTlstmSessionInvalidServerCertificates object is incremented. If the session cannot be opened for any reason at all, including cryptographic verification failures and snmpTlstmCertToTSNTable lookup failures, then the snmpTlstmSessionOpenErrors counter is incremented and processing stops.

6) The TLSTM-specific session identifier (tlstmSessionID) is set in the tmSessionID of the tmStateReference passed to the TLS Transport Model to indicate that the session has been established successfully and to point to a specific (D)TLS connection for future use. The tlstmSessionID is also stored in the LCD for later lookup during processing of incoming messages (Section 5.1.2).

### 5.3.2. Accepting a Session as a Server

A (D)TLS server should accept new session connections from any client for which it is able to verify the client's credentials. This is done by authenticating the client's presented certificate through a certificate path validation process (e.g., [RFC5280]) or through certificate fingerprint verification using fingerprints configured in the snmpTlstmCertToTSNTable. Afterward, the server will determine the identity of the remote entity using the following procedures.

The (D)TLS server identifies the authenticated identity from the (D)TLS client's principal certificate using configuration information from the snmpTlstmCertToTSNTable mapping table. The (D)TLS server MUST request and expect a certificate from the client and MUST NOT accept SNMP messages over the (D)TLS connection until the client has sent a certificate and it has been authenticated. The resulting derived tmSecurityName is recorded in the tmStateReference cache as tmSecurityName. The details of the lookup process are fully described in the DESCRIPTION clause of the snmpTlstmCertToTSNTable MIB object. If any verification fails in any way (for example, because of failures in cryptographic verification or because of the lack of an appropriate row in the snmpTlstmCertToTSNTable), then the session establishment MUST fail, and the snmpTlstmSessionInvalidClientCertificates object is incremented. If the session cannot be opened for any reason at all, including cryptographic verification failures, then the snmpTlstmSessionOpenErrors counter is incremented and processing stops.

Servers that wish to support multiple principals at a particular port SHOULD make use of a (D)TLS extension that allows server-side principal selection like the Server Name Indication extension defined in Section 3.1 of [RFC4366]. Supporting this will allow, for example, sending notifications to a specific principal at a given TCP or UDP port.

## 5.4. Closing a Session

The TLS Transport Model provides the following primitive to close a session:

```
statusInformation =
closeSession(
IN  tmSessionID        -- session ID of the session to be closed
)
```

The following describes the procedure to follow to close a session between a client and server. This process is followed by any SNMP engine closing the corresponding SNMP session.

1) Increment either the snmpTlstmSessionClientCloses or the snmpTlstmSessionServerCloses counter as appropriate.

2) Look up the session using the tmSessionID.

3) If there is no open session associated with the tmSessionID, then closeSession processing is completed.

4) Have (D)TLS close the specified connection. This MUST include sending a close_notify TLS Alert to inform the other side that session cleanup may be performed.

## 6. MIB Module Overview

This MIB module provides management of the TLS Transport Model. It defines needed textual conventions, statistical counters, notifications, and configuration infrastructure necessary for session establishment. Example usage of the configuration tables can be found in Appendix A.

### 6.1. 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.

### 6.2. Textual Conventions

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

This module defines the following new Textual Conventions:

- A new TransportAddress format for describing (D)TLS connection addressing requirements.

- A certificate fingerprint allowing MIB module objects to generically refer to a stored X.509 certificate using a cryptographic hash as a reference pointer.

### 6.3. Statistical Counters

The SNMP-TLS-TM-MIB defines counters that provide network management stations with information about session usage and potential errors that a device may be experiencing.

### 6.4. Configuration Tables

The SNMP-TLS-TM-MIB defines configuration tables that an administrator can use for configuring a device for sending and receiving SNMP messages over (D)TLS. In particular, there are MIB tables that extend the SNMP-TARGET-MIB for configuring (D)TLS certificate usage and a MIB table for mapping incoming (D)TLS client certificates to SNMPv3 tmSecurityNames.

#### 6.4.1. Notifications

The SNMP-TLS-TM-MIB defines notifications to alert management stations when a (D)TLS connection fails because a server's presented certificate did not meet an expected value (snmpTlstmServerCertificateUnknown) or because cryptographic validation failed (snmpTlstmServerInvalidCertificate).

### 6.5. Relationship to Other MIB Modules

Some management objects defined in other MIB modules are applicable to an entity implementing the TLS Transport Model. In particular, it is assumed that an entity implementing the SNMP-TLS-TM-MIB will implement the SNMPv2-MIB [RFC3418], the SNMP-FRAMEWORK-MIB [RFC3411], the SNMP-TARGET-MIB [RFC3413], the SNMP-NOTIFICATION-MIB [RFC3413], and the SNMP-VIEW-BASED-ACM-MIB [RFC3415].

The SNMP-TLS-TM-MIB module contained in this document is for managing TLS Transport Model information.

#### 6.5.1. MIB Modules Required for IMPORTS

The SNMP-TLS-TM-MIB module imports items from SNMPv2-SMI [RFC2578], SNMPv2-TC [RFC2579], SNMP-FRAMEWORK-MIB [RFC3411], SNMP-TARGET-MIB [RFC3413], and SNMPv2-CONF [RFC2580].

## 7. MIB Module Definition

```
SNMP-TLS-TM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    OBJECT-IDENTITY, mib-2, snmpDomains,
    Counter32, Unsigned32, Gauge32, NOTIFICATION-TYPE
      FROM SNMPv2-SMI                 -- RFC 2578 or any update thereof
    TEXTUAL-CONVENTION, TimeStamp, RowStatus, StorageType,
    AutonomousType
      FROM SNMPv2-TC                  -- RFC 2579 or any update thereof
    MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP
      FROM SNMPv2-CONF                -- RFC 2580 or any update thereof
    SnmpAdminString
      FROM SNMP-FRAMEWORK-MIB         -- RFC 3411 or any update thereof
    snmpTargetParamsName, snmpTargetAddrName
      FROM SNMP-TARGET-MIB            -- RFC 3413 or any update thereof
    ;

snmpTlstmMIB MODULE-IDENTITY
    LAST-UPDATED "201107190000Z"
```

```
    ORGANIZATION "ISMS Working Group"
    CONTACT-INFO "WG-EMail:   isms@lists.ietf.org
                  Subscribe:  isms-request@lists.ietf.org

                  Chairs:
                     Juergen Schoenwaelder
                     Jacobs University Bremen
                     Campus Ring 1
                     28725 Bremen
                     Germany
                     +49 421 200-3587
                     j.schoenwaelder@jacobs-university.de

                     Russ Mundy
                     SPARTA, Inc.
                     7110 Samuel Morse Drive
                     Columbia, MD  21046
                     USA

                  Editor:
                     Wes Hardaker
                     SPARTA, Inc.
                     P.O. Box 382
                     Davis, CA  95617
                     USA
                     ietf@hardakers.net
                  "

    DESCRIPTION  "
        The TLS Transport Model MIB

        Copyright (c) 2010-2011 IETF Trust and the persons identified
        as authors of the code.  All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (http://trustee.ietf.org/license-info)."

       REVISION     "201107190000Z"
       DESCRIPTION  "This version of this MIB module is part of
                     RFC 6353; see the RFC itself for full legal
                     notices.  The only change was to introduce
                     new wording to reflect require changes for
                     IDNA addresses in the SnmpTLSAddress TC."
```

```
        REVISION     "201005070000Z"
        DESCRIPTION  "This version of this MIB module is part of
                      RFC 5953; see the RFC itself for full legal
                      notices."

    ::= { mib-2 198 }

-- ************************************************
-- subtrees of the SNMP-TLS-TM-MIB
-- ************************************************

snmpTlstmNotifications OBJECT IDENTIFIER ::= { snmpTlstmMIB 0 }
snmpTlstmIdentities    OBJECT IDENTIFIER ::= { snmpTlstmMIB 1 }
snmpTlstmObjects       OBJECT IDENTIFIER ::= { snmpTlstmMIB 2 }
snmpTlstmConformance   OBJECT IDENTIFIER ::= { snmpTlstmMIB 3 }

-- ************************************************
-- snmpTlstmObjects - Objects
-- ************************************************

snmpTLSTCPDomain OBJECT-IDENTITY
    STATUS      current
    DESCRIPTION
        "The SNMP over TLS via TCP transport domain.  The
        corresponding transport address is of type SnmpTLSAddress.

        The securityName prefix to be associated with the
        snmpTLSTCPDomain is 'tls'.  This prefix may be used by
        security models or other components to identify which secure
        transport infrastructure authenticated a securityName."
    REFERENCE
      "RFC 2579: Textual Conventions for SMIv2"
    ::= { snmpDomains 8 }

snmpDTLSUDPDomain OBJECT-IDENTITY
    STATUS      current
    DESCRIPTION
        "The SNMP over DTLS via UDP transport domain.  The
        corresponding transport address is of type SnmpTLSAddress.

        The securityName prefix to be associated with the
        snmpDTLSUDPDomain is 'dtls'.  This prefix may be used by
        security models or other components to identify which secure
        transport infrastructure authenticated a securityName."
    REFERENCE
      "RFC 2579: Textual Conventions for SMIv2"
    ::= { snmpDomains 9 }
```

```
SnmpTLSAddress ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1a"
    STATUS       current
    DESCRIPTION
        "Represents an IPv4 address, an IPv6 address, or a
         US-ASCII-encoded hostname and port number.

        An IPv4 address must be in dotted decimal format followed by a
        colon ':' (US-ASCII character 0x3A) and a decimal port number
        in US-ASCII.

        An IPv6 address must be a colon-separated format (as described
        in RFC 5952), surrounded by square brackets ('[', US-ASCII
        character 0x5B, and ']', US-ASCII character 0x5D), followed by
        a colon ':' (US-ASCII character 0x3A) and a decimal port number
        in US-ASCII.

        A hostname is always in US-ASCII (as per RFC 1123);
        internationalized hostnames are encoded as A-labels as specified
        in  RFC 5890.  The hostname is followed by a
        colon ':' (US-ASCII character 0x3A) and a decimal port number
        in US-ASCII.  The name SHOULD be fully qualified whenever
        possible.

        Values of this textual convention may not be directly usable
        as transport-layer addressing information, and may require
        run-time resolution.  As such, applications that write them
        must be prepared for handling errors if such values are not
        supported, or cannot be resolved (if resolution occurs at the
        time of the management operation).

        The DESCRIPTION clause of TransportAddress objects that may
        have SnmpTLSAddress values must fully describe how (and
        when) such names are to be resolved to IP addresses and vice
        versa.

        This textual convention SHOULD NOT be used directly in object
        definitions since it restricts addresses to a specific
        format.  However, if it is used, it MAY be used either on its
        own or in conjunction with TransportAddressType or
        TransportDomain as a pair.

        When this textual convention is used as a syntax of an index
        object, there may be issues with the limit of 128
        sub-identifiers specified in SMIv2 (STD 58).  It is RECOMMENDED
        that all MIB documents using this textual convention make
        explicit any limitations on index component lengths that
        management software must observe.  This may be done either by
```

```
        including SIZE constraints on the index components or by
        specifying applicable constraints in the conceptual row
        DESCRIPTION clause or in the surrounding documentation."
    REFERENCE
      "RFC 1123: Requirements for Internet Hosts - Application and
                 Support
       RFC 5890: Internationalized Domain Names for Applications (IDNA):
                 Definitions and Document Framework
       RFC 5952: A Recommendation for IPv6 Address Text Representation
      "
    SYNTAX       OCTET STRING (SIZE (1..255))

SnmpTLSFingerprint ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1x:1x"
    STATUS       current
    DESCRIPTION
       "A fingerprint value that can be used to uniquely reference
       other data of potentially arbitrary length.

       An SnmpTLSFingerprint value is composed of a 1-octet hashing
       algorithm identifier followed by the fingerprint value.  The
       octet value encoded is taken from the IANA TLS HashAlgorithm
       Registry (RFC 5246).  The remaining octets are filled using the
       results of the hashing algorithm.

       This TEXTUAL-CONVENTION allows for a zero-length (blank)
       SnmpTLSFingerprint value for use in tables where the
       fingerprint value may be optional.  MIB definitions or
       implementations may refuse to accept a zero-length value as
       appropriate."
       REFERENCE "RFC 5246: The Transport Layer
                  Security (TLS) Protocol Version 1.2
                  http://www.iana.org/assignments/tls-parameters/
       "
    SYNTAX OCTET STRING (SIZE (0..255))

-- Identities for use in the snmpTlstmCertToTSNTable

snmpTlstmCertToTSNMIdentities OBJECT IDENTIFIER
    ::= { snmpTlstmIdentities 1 }

snmpTlstmCertSpecified OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION  "Directly specifies the tmSecurityName to be used for
                  this certificate.  The value of the tmSecurityName
                  to use is specified in the snmpTlstmCertToTSNData
                  column.  The snmpTlstmCertToTSNData column must
                  contain a non-zero length SnmpAdminString compliant
```

```
                value or the mapping described in this row must be
                considered a failure."
    ::= { snmpTlstmCertToTSNMIdentities 1 }

snmpTlstmCertSANRFC822Name OBJECT-IDENTITY
    STATUS       current
    DESCRIPTION  "Maps a subjectAltName's rfc822Name to a
                tmSecurityName.  The local part of the rfc822Name is
                passed unaltered but the host-part of the name must
                be passed in lowercase.  This mapping results in a
                1:1 correspondence between equivalent subjectAltName
                rfc822Name values and tmSecurityName values except
                that the host-part of the name MUST be passed in
                lowercase.

                Example rfc822Name Field:  FooBar@Example.COM
                is mapped to tmSecurityName: FooBar@example.com."
    ::= { snmpTlstmCertToTSNMIdentities 2 }

snmpTlstmCertSANDNSName OBJECT-IDENTITY
    STATUS       current
    DESCRIPTION  "Maps a subjectAltName's dNSName to a
                tmSecurityName after first converting it to all
                lowercase (RFC 5280 does not specify converting to
                lowercase so this involves an extra step).  This
                mapping results in a 1:1 correspondence between
                subjectAltName dNSName values and the tmSecurityName
                values."
    REFERENCE "RFC 5280 - Internet X.509 Public Key Infrastructure
                         Certificate and Certificate Revocation
                         List (CRL) Profile."
    ::= { snmpTlstmCertToTSNMIdentities 3 }

snmpTlstmCertSANIpAddress OBJECT-IDENTITY
    STATUS       current
    DESCRIPTION  "Maps a subjectAltName's iPAddress to a
                tmSecurityName by transforming the binary encoded
                address as follows:

                1) for IPv4, the value is converted into a
                   decimal-dotted quad address (e.g., '192.0.2.1').

                2) for IPv6 addresses, the value is converted into a
                   32-character all lowercase hexadecimal string
                   without any colon separators.
```

```
                    This mapping results in a 1:1 correspondence between
                    subjectAltName iPAddress values and the
                    tmSecurityName values.

                    The resulting length of an encoded IPv6 address is
                    the maximum length supported by the View-Based
                    Access Control Model (VACM).  Using both the
                    Transport Security Model's support for transport
                    prefixes (see the SNMP-TSM-MIB's
                    snmpTsmConfigurationUsePrefix object for details)
                    will result in securityName lengths that exceed what
                    VACM can handle."
    ::= { snmpTlstmCertToTSNMIdentities 4 }

snmpTlstmCertSANAny OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION  "Maps any of the following fields using the
                  corresponding mapping algorithms:

                  |------------+----------------------------|
                  | Type       | Algorithm                  |
                  |------------+----------------------------|
                  | rfc822Name | snmpTlstmCertSANRFC822Name |
                  | dNSName    | snmpTlstmCertSANDNSName    |
                  | iPAddress  | snmpTlstmCertSANIpAddress  |
                  |------------+----------------------------|

                  The first matching subjectAltName value found in the
                  certificate of the above types MUST be used when
                  deriving the tmSecurityName.  The mapping algorithm
                  specified in the 'Algorithm' column MUST be used to
                  derive the tmSecurityName.

                  This mapping results in a 1:1 correspondence between
                  subjectAltName values and tmSecurityName values.  The
                  three sub-mapping algorithms produced by this
                  combined algorithm cannot produce conflicting
                  results between themselves."
    ::= { snmpTlstmCertToTSNMIdentities 5 }

snmpTlstmCertCommonName OBJECT-IDENTITY
    STATUS        current

    DESCRIPTION  "Maps a certificate's CommonName to a tmSecurityName
                  after converting it to a UTF-8 encoding.  The usage
                  of CommonNames is deprecated and users are
                  encouraged to use subjectAltName mapping methods
                  instead.  This mapping results in a 1:1
```

```
                  correspondence between certificate CommonName values
                  and tmSecurityName values."
    ::= { snmpTlstmCertToTSNMIdentities 6 }

-- The snmpTlstmSession Group

snmpTlstmSession           OBJECT IDENTIFIER ::= { snmpTlstmObjects 1 }

snmpTlstmSessionOpens  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
       "The number of times an openSession() request has been executed
       as a (D)TLS client, regardless of whether it succeeded or
       failed."
    ::= { snmpTlstmSession 1 }

snmpTlstmSessionClientCloses  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times a closeSession() request has been
        executed as a (D)TLS client, regardless of whether it
        succeeded or failed."
    ::= { snmpTlstmSession 2 }

snmpTlstmSessionOpenErrors  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times an openSession() request failed to open a
        session as a (D)TLS client, for any reason."
    ::= { snmpTlstmSession 3 }

snmpTlstmSessionAccepts  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
       "The number of times a (D)TLS server has accepted a new
       connection from a client and has received at least one SNMP
       message through it."
    ::= { snmpTlstmSession 4 }
```

```
snmpTlstmSessionServerCloses  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times a closeSession() request has been
        executed as a (D)TLS server, regardless of whether it
        succeeded or failed."
    ::= { snmpTlstmSession 5 }

snmpTlstmSessionNoSessions  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times an outgoing message was dropped because
        the session associated with the passed tmStateReference was no
        longer (or was never) available."
    ::= { snmpTlstmSession 6 }

snmpTlstmSessionInvalidClientCertificates OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times an incoming session was not established
        on a (D)TLS server because the presented client certificate
        was invalid.  Reasons for invalidation include, but are not
        limited to, cryptographic validation failures or lack of a
        suitable mapping row in the snmpTlstmCertToTSNTable."
    ::= { snmpTlstmSession 7 }

snmpTlstmSessionUnknownServerCertificate OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times an outgoing session was not established
         on a (D)TLS client because the server certificate presented
         by an SNMP over (D)TLS server was invalid because no
         configured fingerprint or Certification Authority (CA) was
         acceptable to validate it.
         This may result because there was no entry in the
         snmpTlstmAddrTable or because no path could be found to a
         known CA."
    ::= { snmpTlstmSession 8 }
```

```
snmpTlstmSessionInvalidServerCertificates OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times an outgoing session was not established
         on a (D)TLS client because the server certificate presented
         by an SNMP over (D)TLS server could not be validated even if
         the fingerprint or expected validation path was known.  That
         is, a cryptographic validation error occurred during
         certificate validation processing.

        Reasons for invalidation include, but are not
        limited to, cryptographic validation failures."
    ::= { snmpTlstmSession 9 }

snmpTlstmSessionInvalidCaches OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of outgoing messages dropped because the
        tmStateReference referred to an invalid cache."
    ::= { snmpTlstmSession 10 }

-- Configuration Objects

snmpTlstmConfig             OBJECT IDENTIFIER ::= { snmpTlstmObjects 2 }

-- Certificate mapping

snmpTlstmCertificateMapping OBJECT IDENTIFIER ::= { snmpTlstmConfig 1 }

snmpTlstmCertToTSNCount OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "A count of the number of entries in the
        snmpTlstmCertToTSNTable."
    ::= { snmpTlstmCertificateMapping 1 }

snmpTlstmCertToTSNTableLastChanged OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
```

```
    DESCRIPTION
        "The value of sysUpTime.0 when the snmpTlstmCertToTSNTable was
        last modified through any means, or 0 if it has not been
        modified since the command responder was started."
    ::= { snmpTlstmCertificateMapping 2 }

snmpTlstmCertToTSNTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF SnmpTlstmCertToTSNEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table is used by a (D)TLS server to map the (D)TLS
        client's presented X.509 certificate to a tmSecurityName.

        On an incoming (D)TLS/SNMP connection, the client's presented
        certificate must either be validated based on an established
        trust anchor, or it must directly match a fingerprint in this
        table.  This table does not provide any mechanisms for
        configuring the trust anchors; the transfer of any needed
        trusted certificates for path validation is expected to occur
        through an out-of-band transfer.

        Once the certificate has been found acceptable (either by path
        validation or directly matching a fingerprint in this table),
        this table is consulted to determine the appropriate
        tmSecurityName to identify with the remote connection.  This
        is done by considering each active row from this table in
        prioritized order according to its snmpTlstmCertToTSNID value.
        Each row's snmpTlstmCertToTSNFingerprint value determines
        whether the row is a match for the incoming connection:

            1) If the row's snmpTlstmCertToTSNFingerprint value
               identifies the presented certificate, then consider the
               row as a successful match.

            2) If the row's snmpTlstmCertToTSNFingerprint value
               identifies a locally held copy of a trusted CA
               certificate and that CA certificate was used to
               validate the path to the presented certificate, then
               consider the row as a successful match.

        Once a matching row has been found, the
        snmpTlstmCertToTSNMapType value can be used to determine how
        the tmSecurityName to associate with the session should be
        determined.  See the snmpTlstmCertToTSNMapType column's
        DESCRIPTION for details on determining the tmSecurityName
        value.  If it is impossible to determine a tmSecurityName from
        the row's data combined with the data presented in the
```

```
        certificate, then additional rows MUST be searched looking for
        another potential match.  If a resulting tmSecurityName mapped
        from a given row is not compatible with the needed
        requirements of a tmSecurityName (e.g., VACM imposes a
        32-octet-maximum length and the certificate derived
        securityName could be longer), then it must be considered an
        invalid match and additional rows MUST be searched looking for
        another potential match.

        If no matching and valid row can be found, the connection MUST
        be closed and SNMP messages MUST NOT be accepted over it.

        Missing values of snmpTlstmCertToTSNID are acceptable and
        implementations should continue to the next highest numbered
        row.  It is recommended that administrators skip index values
        to leave room for the insertion of future rows (for example,
        use values of 10 and 20 when creating initial rows).

        Users are encouraged to make use of certificates with
        subjectAltName fields that can be used as tmSecurityNames so
        that a single root CA certificate can allow all child
        certificate's subjectAltName to map directly to a
        tmSecurityName via a 1:1 transformation.  However, this table
        is flexible to allow for situations where existing deployed
        certificate infrastructures do not provide adequate
        subjectAltName values for use as tmSecurityNames.
        Certificates may also be mapped to tmSecurityNames using the
        CommonName portion of the Subject field.  However, the usage
        of the CommonName field is deprecated and thus this usage is
        NOT RECOMMENDED.  Direct mapping from each individual
        certificate fingerprint to a tmSecurityName is also possible
        but requires one entry in the table per tmSecurityName and
        requires more management operations to completely configure a
        device."
    ::= { snmpTlstmCertificateMapping 3 }

snmpTlstmCertToTSNEntry OBJECT-TYPE
    SYNTAX      SnmpTlstmCertToTSNEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A row in the snmpTlstmCertToTSNTable that specifies a mapping
        for an incoming (D)TLS certificate to a tmSecurityName to use
        for a connection."
    INDEX   { snmpTlstmCertToTSNID }
    ::= { snmpTlstmCertToTSNTable 1 }
```

```
SnmpTlstmCertToTSNEntry ::= SEQUENCE {
    snmpTlstmCertToTSNID           Unsigned32,
    snmpTlstmCertToTSNFingerprint  SnmpTLSFingerprint,
    snmpTlstmCertToTSNMapType      AutonomousType,
    snmpTlstmCertToTSNData         OCTET STRING,
    snmpTlstmCertToTSNStorageType  StorageType,
    snmpTlstmCertToTSNRowStatus    RowStatus
}

snmpTlstmCertToTSNID OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A unique, prioritized index for the given entry.  Lower
        numbers indicate a higher priority."
    ::= { snmpTlstmCertToTSNEntry 1 }

snmpTlstmCertToTSNFingerprint OBJECT-TYPE
    SYNTAX      SnmpTLSFingerprint (SIZE(1..255))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "A cryptographic hash of an X.509 certificate.  The results of
        a successful matching fingerprint to either the trusted CA in
        the certificate validation path or to the certificate itself
        is dictated by the snmpTlstmCertToTSNMapType column."
    ::= { snmpTlstmCertToTSNEntry 2 }

snmpTlstmCertToTSNMapType OBJECT-TYPE
    SYNTAX      AutonomousType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "Specifies the mapping type for deriving a tmSecurityName from
        a certificate.  Details for mapping of a particular type SHALL
        be specified in the DESCRIPTION clause of the OBJECT-IDENTITY
        that describes the mapping.  If a mapping succeeds it will
        return a tmSecurityName for use by the TLSTM model and
        processing stops.

        If the resulting mapped value is not compatible with the
        needed requirements of a tmSecurityName (e.g., VACM imposes a
        32-octet-maximum length and the certificate derived
        securityName could be longer), then future rows MUST be
        searched for additional snmpTlstmCertToTSNFingerprint matches
        to look for a mapping that succeeds.
```

```
        Suitable values for assigning to this object that are defined
        within the SNMP-TLS-TM-MIB can be found in the
        snmpTlstmCertToTSNMIdentities portion of the MIB tree."
    DEFVAL { snmpTlstmCertSpecified }
    ::= { snmpTlstmCertToTSNEntry 3 }

snmpTlstmCertToTSNData OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE(0..1024))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "Auxiliary data used as optional configuration information for
        a given mapping specified by the snmpTlstmCertToTSNMapType
        column.  Only some mapping systems will make use of this
        column.  The value in this column MUST be ignored for any
        mapping type that does not require data present in this
        column."
    DEFVAL { "" }
    ::= { snmpTlstmCertToTSNEntry 4 }

snmpTlstmCertToTSNStorageType OBJECT-TYPE
    SYNTAX       StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The storage type for this conceptual row.  Conceptual rows
        having the value 'permanent' need not allow write-access to
        any columnar objects in the row."
    DEFVAL      { nonVolatile }
    ::= { snmpTlstmCertToTSNEntry 5 }

snmpTlstmCertToTSNRowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The status of this conceptual row.  This object may be used
        to create or remove rows from this table.

        To create a row in this table, an administrator must set this
        object to either createAndGo(4) or createAndWait(5).

        Until instances of all corresponding columns are appropriately
        configured, the value of the corresponding instance of the
        snmpTlstmParamsRowStatus column is notReady(3).

        In particular, a newly created row cannot be made active until
        the corresponding snmpTlstmCertToTSNFingerprint,
```

```
        snmpTlstmCertToTSNMapType, and snmpTlstmCertToTSNData columns
        have been set.

        The following objects may not be modified while the
        value of this object is active(1):
            - snmpTlstmCertToTSNFingerprint
            - snmpTlstmCertToTSNMapType
            - snmpTlstmCertToTSNData
        An attempt to set these objects while the value of
        snmpTlstmParamsRowStatus is active(1) will result in
        an inconsistentValue error."
    ::= { snmpTlstmCertToTSNEntry 6 }

-- Maps tmSecurityNames to certificates for use by the SNMP-TARGET-MIB

snmpTlstmParamsCount OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "A count of the number of entries in the snmpTlstmParamsTable."
    ::= { snmpTlstmCertificateMapping 4 }

snmpTlstmParamsTableLastChanged OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of sysUpTime.0 when the snmpTlstmParamsTable
        was last modified through any means, or 0 if it has not been
        modified since the command responder was started."
    ::= { snmpTlstmCertificateMapping 5 }

snmpTlstmParamsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF SnmpTlstmParamsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table is used by a (D)TLS client when a (D)TLS
        connection is being set up using an entry in the
        SNMP-TARGET-MIB.  It extends the SNMP-TARGET-MIB's
        snmpTargetParamsTable with a fingerprint of a certificate to
        use when establishing such a (D)TLS connection."
    ::= { snmpTlstmCertificateMapping 6 }

snmpTlstmParamsEntry OBJECT-TYPE
    SYNTAX      SnmpTlstmParamsEntry
    MAX-ACCESS  not-accessible
```

```
    STATUS      current
    DESCRIPTION
        "A conceptual row containing a fingerprint hash of a locally
        held certificate for a given snmpTargetParamsEntry.  The
        values in this row should be ignored if the connection that
        needs to be established, as indicated by the SNMP-TARGET-MIB
        infrastructure, is not a certificate and (D)TLS based
        connection.  The connection SHOULD NOT be established if the
        certificate fingerprint stored in this entry does not point to
        a valid locally held certificate or if it points to an
        unusable certificate (such as might happen when the
        certificate's expiration date has been reached)."
    INDEX    { IMPLIED snmpTargetParamsName }
    ::= { snmpTlstmParamsTable 1 }

SnmpTlstmParamsEntry ::= SEQUENCE {
    snmpTlstmParamsClientFingerprint SnmpTLSFingerprint,
    snmpTlstmParamsStorageType       StorageType,
    snmpTlstmParamsRowStatus         RowStatus
}

snmpTlstmParamsClientFingerprint OBJECT-TYPE
    SYNTAX      SnmpTLSFingerprint
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This object stores the hash of the public portion of a
        locally held X.509 certificate.  The X.509 certificate, its
        public key, and the corresponding private key will be used
        when initiating a (D)TLS connection as a (D)TLS client."
    ::= { snmpTlstmParamsEntry 1 }

snmpTlstmParamsStorageType OBJECT-TYPE
    SYNTAX       StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The storage type for this conceptual row.  Conceptual rows
        having the value 'permanent' need not allow write-access to
        any columnar objects in the row."
    DEFVAL      { nonVolatile }
    ::= { snmpTlstmParamsEntry 2 }

snmpTlstmParamsRowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
```

```
        "The status of this conceptual row.  This object may be used
        to create or remove rows from this table.

        To create a row in this table, an administrator must set this
        object to either createAndGo(4) or createAndWait(5).

        Until instances of all corresponding columns are appropriately
        configured, the value of the corresponding instance of the
        snmpTlstmParamsRowStatus column is notReady(3).

        In particular, a newly created row cannot be made active until
        the corresponding snmpTlstmParamsClientFingerprint column has
        been set.

        The snmpTlstmParamsClientFingerprint object may not be modified
        while the value of this object is active(1).

        An attempt to set these objects while the value of
        snmpTlstmParamsRowStatus is active(1) will result in
        an inconsistentValue error."
    ::= { snmpTlstmParamsEntry 3 }

snmpTlstmAddrCount OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "A count of the number of entries in the snmpTlstmAddrTable."
    ::= { snmpTlstmCertificateMapping 7 }

snmpTlstmAddrTableLastChanged OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of sysUpTime.0 when the snmpTlstmAddrTable
        was last modified through any means, or 0 if it has not been
        modified since the command responder was started."
    ::= { snmpTlstmCertificateMapping 8 }

snmpTlstmAddrTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF SnmpTlstmAddrEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table is used by a (D)TLS client when a (D)TLS
        connection is being set up using an entry in the
        SNMP-TARGET-MIB.  It extends the SNMP-TARGET-MIB's
```

snmpTargetAddrTable so that the client can verify that the correct server has been reached. This verification can use either a certificate fingerprint, or an identity authenticated via certification path validation.

If there is an active row in this table corresponding to the entry in the SNMP-TARGET-MIB that was used to establish the connection, and the row's snmpTlstmAddrServerFingerprint column has non-empty value, then the server's presented certificate is compared with the snmpTlstmAddrServerFingerprint value (and the snmpTlstmAddrServerIdentity column is ignored). If the fingerprint matches, the verification has succeeded. If the fingerprint does not match, then the connection MUST be closed.

If the server's presented certificate has passed certification path validation [RFC5280] to a configured trust anchor, and an active row exists with a zero-length snmpTlstmAddrServerFingerprint value, then the snmpTlstmAddrServerIdentity column contains the expected host name. This expected host name is then compared against the server's certificate as follows:

- Implementations MUST support matching the expected host name against a dNSName in the subjectAltName extension field and MAY support checking the name against the CommonName portion of the subject distinguished name.

- The '*' (ASCII 0x2a) wildcard character is allowed in the dNSName of the subjectAltName extension (and in common name, if used to store the host name), but only as the left-most (least significant) DNS label in that value. This wildcard matches any left-most DNS label in the server name. That is, the subject *.example.com matches the server names a.example.com and b.example.com, but does not match example.com or a.b.example.com. Implementations MUST support wildcards in certificates as specified above, but MAY provide a configuration option to disable them.

- If the locally configured name is an internationalized domain name, conforming implementations MUST convert it to the ASCII Compatible Encoding (ACE) format for performing comparisons, as specified in Section 7 of [RFC5280].

If the expected host name fails these conditions then the connection MUST be closed.

```
        If there is no row in this table corresponding to the entry
        in the SNMP-TARGET-MIB and the server can be authorized by
        another, implementation-dependent means, then the connection
        MAY still proceed."

    ::= { snmpTlstmCertificateMapping 9 }

snmpTlstmAddrEntry OBJECT-TYPE
    SYNTAX      SnmpTlstmAddrEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A conceptual row containing a copy of a certificate's
        fingerprint for a given snmpTargetAddrEntry.  The values in
        this row should be ignored if the connection that needs to be
        established, as indicated by the SNMP-TARGET-MIB
        infrastructure, is not a (D)TLS based connection.  If an
        snmpTlstmAddrEntry exists for a given snmpTargetAddrEntry, then
        the presented server certificate MUST match or the connection
        MUST NOT be established.  If a row in this table does not
        exist to match an snmpTargetAddrEntry row, then the connection
        SHOULD still proceed if some other certificate validation path
        algorithm (e.g., RFC 5280) can be used."
    INDEX    { IMPLIED snmpTargetAddrName }
    ::= { snmpTlstmAddrTable 1 }

SnmpTlstmAddrEntry ::= SEQUENCE {
    snmpTlstmAddrServerFingerprint    SnmpTLSFingerprint,
    snmpTlstmAddrServerIdentity       SnmpAdminString,
    snmpTlstmAddrStorageType          StorageType,
    snmpTlstmAddrRowStatus            RowStatus
}

snmpTlstmAddrServerFingerprint OBJECT-TYPE
    SYNTAX      SnmpTLSFingerprint
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "A cryptographic hash of a public X.509 certificate.  This
        object should store the hash of the public X.509 certificate
        that the remote server should present during the (D)TLS
        connection setup.  The fingerprint of the presented
        certificate and this hash value MUST match exactly or the
        connection MUST NOT be established."
    DEFVAL { "" }
    ::= { snmpTlstmAddrEntry 1 }
```

```
snmpTlstmAddrServerIdentity OBJECT-TYPE
    SYNTAX      SnmpAdminString
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The reference identity to check against the identity
        presented by the remote system."
    DEFVAL { "" }
    ::= { snmpTlstmAddrEntry 2 }

snmpTlstmAddrStorageType OBJECT-TYPE
    SYNTAX       StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The storage type for this conceptual row.  Conceptual rows
        having the value 'permanent' need not allow write-access to
        any columnar objects in the row."
    DEFVAL      { nonVolatile }
    ::= { snmpTlstmAddrEntry 3 }


snmpTlstmAddrRowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The status of this conceptual row.  This object may be used
        to create or remove rows from this table.

        To create a row in this table, an administrator must set this
        object to either createAndGo(4) or createAndWait(5).

        Until instances of all corresponding columns are
        appropriately configured, the value of the
        corresponding instance of the snmpTlstmAddrRowStatus
        column is notReady(3).

        In particular, a newly created row cannot be made active until
        the corresponding snmpTlstmAddrServerFingerprint column has been
        set.

        Rows MUST NOT be active if the snmpTlstmAddrServerFingerprint
        column is blank and the snmpTlstmAddrServerIdentity is set to
        '*' since this would insecurely accept any presented
        certificate.
```

```
        The snmpTlstmAddrServerFingerprint object may not be modified
        while the value of this object is active(1).

        An attempt to set these objects while the value of
        snmpTlstmAddrRowStatus is active(1) will result in
        an inconsistentValue error."
    ::= { snmpTlstmAddrEntry 4 }

-- ************************************************
--  snmpTlstmNotifications - Notifications Information
-- ************************************************

snmpTlstmServerCertificateUnknown NOTIFICATION-TYPE
    OBJECTS { snmpTlstmSessionUnknownServerCertificate }
    STATUS  current
    DESCRIPTION
        "Notification that the server certificate presented by an SNMP
         over (D)TLS server was invalid because no configured
         fingerprint or CA was acceptable to validate it.  This may be
         because there was no entry in the snmpTlstmAddrTable or
         because no path could be found to known Certification
         Authority.

         To avoid notification loops, this notification MUST NOT be
         sent to servers that themselves have triggered the
         notification."
    ::= { snmpTlstmNotifications 1 }

snmpTlstmServerInvalidCertificate NOTIFICATION-TYPE
    OBJECTS { snmpTlstmAddrServerFingerprint,
              snmpTlstmSessionInvalidServerCertificates}
    STATUS  current
    DESCRIPTION
        "Notification that the server certificate presented by an SNMP
         over (D)TLS server could not be validated even if the
         fingerprint or expected validation path was known.  That is, a
         cryptographic validation error occurred during certificate
         validation processing.

         To avoid notification loops, this notification MUST NOT be
         sent to servers that themselves have triggered the
         notification."
    ::= { snmpTlstmNotifications 2 }

-- ************************************************
-- snmpTlstmCompliances - Conformance Information
-- ************************************************
```

```
snmpTlstmCompliances OBJECT IDENTIFIER ::= { snmpTlstmConformance 1 }

snmpTlstmGroups OBJECT IDENTIFIER ::= { snmpTlstmConformance 2 }

-- ************************************************
-- Compliance statements
-- ************************************************

snmpTlstmCompliance MODULE-COMPLIANCE
    STATUS      current
    DESCRIPTION
        "The compliance statement for SNMP engines that support the
        SNMP-TLS-TM-MIB"
    MODULE
        MANDATORY-GROUPS { snmpTlstmStatsGroup,
                           snmpTlstmIncomingGroup,
                           snmpTlstmOutgoingGroup,
                           snmpTlstmNotificationGroup }
    ::= { snmpTlstmCompliances 1 }

-- ************************************************
-- Units of conformance
-- ************************************************
snmpTlstmStatsGroup OBJECT-GROUP
    OBJECTS {
        snmpTlstmSessionOpens,
        snmpTlstmSessionClientCloses,
        snmpTlstmSessionOpenErrors,
        snmpTlstmSessionAccepts,
        snmpTlstmSessionServerCloses,
        snmpTlstmSessionNoSessions,
        snmpTlstmSessionInvalidClientCertificates,
        snmpTlstmSessionUnknownServerCertificate,
        snmpTlstmSessionInvalidServerCertificates,
        snmpTlstmSessionInvalidCaches
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects for maintaining
        statistical information of an SNMP engine that
        implements the SNMP TLS Transport Model."
    ::= { snmpTlstmGroups 1 }

snmpTlstmIncomingGroup OBJECT-GROUP
    OBJECTS {
        snmpTlstmCertToTSNCount,
        snmpTlstmCertToTSNTableLastChanged,
        snmpTlstmCertToTSNFingerprint,
```

```
        snmpTlstmCertToTSNMapType,
        snmpTlstmCertToTSNData,
        snmpTlstmCertToTSNStorageType,
        snmpTlstmCertToTSNRowStatus
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects for maintaining
        incoming connection certificate mappings to
        tmSecurityNames of an SNMP engine that implements the
        SNMP TLS Transport Model."
    ::= { snmpTlstmGroups 2 }

snmpTlstmOutgoingGroup OBJECT-GROUP
    OBJECTS {
        snmpTlstmParamsCount,
        snmpTlstmParamsTableLastChanged,
        snmpTlstmParamsClientFingerprint,
        snmpTlstmParamsStorageType,
        snmpTlstmParamsRowStatus,
        snmpTlstmAddrCount,
        snmpTlstmAddrTableLastChanged,
        snmpTlstmAddrServerFingerprint,
        snmpTlstmAddrServerIdentity,
        snmpTlstmAddrStorageType,
        snmpTlstmAddrRowStatus
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects for maintaining
        outgoing connection certificates to use when opening
        connections as a result of SNMP-TARGET-MIB settings."
    ::= { snmpTlstmGroups 3 }

snmpTlstmNotificationGroup NOTIFICATION-GROUP
    NOTIFICATIONS {
        snmpTlstmServerCertificateUnknown,
        snmpTlstmServerInvalidCertificate
    }
    STATUS current
    DESCRIPTION
        "Notifications"
    ::= { snmpTlstmGroups 4 }

END
```

## 8. Operational Considerations

This section discusses various operational aspects of deploying TLSTM.

### 8.1. Sessions

A session is discussed throughout this document as meaning a security association between two TLSTM instances. State information for the sessions are maintained in each TLSTM implementation and this information is created and destroyed as sessions are opened and closed. A "broken" session (one side up and one side down) can result if one side of a session is brought down abruptly (i.e., reboot, power outage, etc.). Whenever possible, implementations SHOULD provide graceful session termination through the use of TLS disconnect messages. Implementations SHOULD also have a system in place for detecting "broken" sessions through the use of heartbeats [HEARTBEAT] or other detection mechanisms.

Implementations SHOULD limit the lifetime of established sessions depending on the algorithms used for generation of the master session secret, the privacy and integrity algorithms used to protect messages, the environment of the session, the amount of data transferred, and the sensitivity of the data.

### 8.2. Notification Receiver Credential Selection

When an SNMP engine needs to establish an outgoing session for notifications, the snmpTargetParamsTable includes an entry for the snmpTargetParamsSecurityName of the target. Servers that wish to support multiple principals at a particular port SHOULD make use of the Server Name Indication extension defined in Section 3.1 of [RFC4366]. Without the Server Name Indication the receiving SNMP engine (server) will not know which (D)TLS certificate to offer to the client so that the tmSecurityName identity-authentication will be successful.

Another solution is to maintain a one-to-one mapping between certificates and incoming ports for notification receivers. This can be handled at the notification originator by configuring the snmpTargetAddrTable (snmpTargetAddrTDomain and snmpTargetAddrTAddress) and requiring the receiving SNMP engine to monitor multiple incoming static ports based on which principals are capable of receiving notifications.

Implementations MAY also choose to designate a single Notification Receiver Principal to receive all incoming notifications or select an

implementation specific method of selecting a server certificate to present to clients.

### 8.3. contextEngineID Discovery

SNMPv3 requires that an application know the identifier (snmpEngineID) of the remote SNMP protocol engine in order to retrieve or manipulate objects maintained on the remote SNMP entity.

[RFC5343] introduces a well-known localEngineID and a discovery mechanism that can be used to learn the snmpEngineID of a remote SNMP protocol engine. Implementations are RECOMMENDED to support and use the contextEngineID discovery mechanism defined in [RFC5343].

### 8.4. Transport Considerations

This document defines how SNMP messages can be transmitted over the TLS- and DTLS-based protocols. Each of these protocols is additionally based on other transports (TCP and UDP). These two base protocols also have operational considerations that must be taken into consideration when selecting a (D)TLS-based protocol to use such as its performance in degraded or limited networks. It is beyond the scope of this document to summarize the characteristics of these transport mechanisms. Please refer to the base protocol documents for details on messaging considerations with respect to MTU size, fragmentation, performance in lossy networks, etc.

## 9. Security Considerations

This document describes a transport model that permits SNMP to utilize (D)TLS security services. The security threats and how the (D)TLS transport model mitigates these threats are covered in detail throughout this document. Security considerations for DTLS are covered in [RFC4347] and security considerations for TLS are described in Section 11 and Appendices D, E, and F of TLS 1.2 [RFC5246]. When run over a connectionless transport such as UDP, DTLS is more vulnerable to denial-of-service attacks from spoofed IP addresses; see Section 4.2 for details how the cookie exchange is used to address this issue.

### 9.1. Certificates, Authentication, and Authorization

Implementations are responsible for providing a security certificate installation and configuration mechanism. Implementations SHOULD support certificate revocation lists.

(D)TLS provides for authentication of the identity of both the (D)TLS server and the (D)TLS client. Access to MIB objects for the

authenticated principal MUST be enforced by an access control subsystem (e.g., the VACM).

Authentication of the command generator principal's identity is important for use with the SNMP access control subsystem to ensure that only authorized principals have access to potentially sensitive data. The authenticated identity of the command generator principal's certificate is mapped to an SNMP model-independent securityName for use with SNMP access control.

The (D)TLS handshake only provides assurance that the certificate of the authenticated identity has been signed by a configured accepted Certification Authority. (D)TLS has no way to further authorize or reject access based on the authenticated identity. An Access Control Model (such as the VACM) provides access control and authorization of a command generator's requests to a command responder and a notification receiver's authorization to receive Notifications from a notification originator. However, to avoid man-in-the-middle attacks, both ends of the (D)TLS-based connection MUST check the certificate presented by the other side against what was expected. For example, command generators must check that the command responder presented and authenticated itself with an X.509 certificate that was expected. Not doing so would allow an impostor, at a minimum, to present false data, receive sensitive information, and/or provide a false belief that configuration was actually received and acted upon. Authenticating and verifying the identity of the (D)TLS server and the (D)TLS client for all operations ensures the authenticity of the SNMP engine that provides MIB data.

The instructions found in the DESCRIPTION clause of the snmpTlstmCertToTSNTable object must be followed exactly. It is also important that the rows of the table be searched in prioritized order starting with the row containing the lowest numbered snmpTlstmCertToTSNID value.

### 9.2. (D)TLS Security Considerations

This section discusses security considerations specific to the usage of (D)TLS.

#### 9.2.1. TLS Version Requirements

Implementations of TLS typically support multiple versions of the Transport Layer Security protocol as well as the older Secure Sockets Layer (SSL) protocol. Because of known security vulnerabilities, TLSTM clients and servers MUST NOT request, offer, or use SSL 2.0. See Appendix E.2 of [RFC5246] for further details.

#### 9.2.2. Perfect Forward Secrecy

The use of Perfect Forward Secrecy is RECOMMENDED and can be provided by (D)TLS with appropriately selected cipher_suites, as discussed in Appendix F of [RFC5246].

### 9.3. Use with SNMPv1/SNMPv2c Messages

The SNMPv1 and SNMPv2c message processing described in [RFC3584] (BCP 74) always selects the SNMPv1 or SNMPv2c Security Models, respectively. Both of these and the User-based Security Model typically used with SNMPv3 derive the securityName and securityLevel from the SNMP message received, even when the message was received over a secure transport. Access control decisions are therefore made based on the contents of the SNMP message, rather than using the authenticated identity and securityLevel provided by the TLS Transport Model. It is RECOMMENDED that only SNMPv3 messages using the Transport Security Model (TSM) or another secure-transport aware security model be sent over the TLSTM transport.

Using a non-transport-aware Security Model with a secure Transport Model is NOT RECOMMENDED. See [RFC5590], Section 7.1 for additional details on the coexistence of security-aware transports and non-transport-aware security models.

### 9.4. MIB Module Security

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:

- The snmpTlstmParamsTable can be used to change the outgoing X.509 certificate used to establish a (D)TLS connection. Modifications to objects in this table need to be adequately authenticated since modifying the values in this table will have profound impacts to the security of outbound connections from the device. Since knowledge of authorization rules and certificate usage mechanisms may be considered sensitive, protection from disclosure of the SNMP traffic via encryption is also highly recommended.

- The snmpTlstmAddrTable can be used to change the expectations of the certificates presented by a remote (D)TLS server. Modifications to objects in this table need to be adequately authenticated since modifying the values in this table will have

profound impacts to the security of outbound connections from the device. Since knowledge of authorization rules and certificate usage mechanisms may be considered sensitive, protection from disclosure of the SNMP traffic via encryption is also highly recommended.

- The snmpTlstmCertToTSNTable is used to specify the mapping of incoming X.509 certificates to tmSecurityNames, which eventually get mapped to an SNMPv3 securityName. Modifications to objects in this table need to be adequately authenticated since modifying the values in this table will have profound impacts to the security of incoming connections to the device. Since knowledge of authorization rules and certificate usage mechanisms may be considered sensitive, protection from disclosure of the SNMP traffic via encryption is also highly recommended. When this table contains a significant number of rows it may affect the system performance when accepting new (D)TLS connections.

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:

- This MIB contains a collection of counters that monitor the (D)TLS connections being established with a device. Since knowledge of connection and certificate usage mechanisms may be considered sensitive, protection from disclosure of the SNMP traffic via encryption is highly recommended.

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.

## 10. IANA Considerations

IANA has assigned:

1. Two TCP/UDP port numbers from the "Registered Ports" range of the Port Numbers registry, with the following keywords:

| Keyword | Decimal | Description | References |
|---|---|---|---|
| snmptls | 10161/tcp | SNMP-TLS | [RFC6353] |
| snmpdtls | 10161/udp | SNMP-DTLS | [RFC6353] |
| snmptls-trap | 10162/tcp | SNMP-Trap-TLS | [RFC6353] |
| snmpdtls-trap | 10162/udp | SNMP-Trap-DTLS | [RFC6353] |

These are the default ports for receipt of SNMP command messages (snmptls and snmpdtls) and SNMP notification messages (snmptls-trap and snmpdtls-trap) over a TLS Transport Model as defined in this document.

2. An SMI number (8) under snmpDomains for the snmpTLSTCPDomain object identifier

3. An SMI number (9) under snmpDomains for the snmpDTLSUDPDomain object identifier

4. An SMI number (198) under mib-2, for the MIB module in this document

5. "tls" as the corresponding prefix for the snmpTLSTCPDomain in the SNMP Transport Domains registry

6. "dtls" as the corresponding prefix for the snmpDTLSUDPDomain in the SNMP Transport Domains registry

## 11. Acknowledgements

This document closely follows and copies the Secure Shell Transport Model for SNMP documented by David Harrington and Joseph Salowey in [RFC5592].

This document was reviewed by the following people who helped provide useful comments (in alphabetical order): Andy Donati, Pasi Eronen, David Harrington, Jeffrey Hutzelman, Alan Luchuk, Michael Peck, Tom Petch, Randy Presuhn, Ray Purvis, Peter Saint-Andre, Joseph Salowey, Juergen Schoenwaelder, Dave Shield, and Robert Story.

This work was supported in part by the United States Department of Defense. Large portions of this document are based on work by General Dynamics C4 Systems and the following individuals: Brian Baril, Kim Bryant, Dana Deluca, Dan Hanson, Tim Huemiller, John Holzhauer, Colin Hoogeboom, Dave Kornbau, Chris Knaian, Dan Knaul, Charles Limoges, Steve Moccaldi, Gerardo Orlando, and Brandon Yip.

## 12. References

### 12.1. Normative References

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

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

[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.

[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.

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

[RFC3415] Wijnen, B., Presuhn, R., and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3415, December 2002.

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

[RFC3584] Frye, R., Levi, D., Routhier, S., and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", BCP 74, RFC 3584, August 2003.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5590] Harrington, D. and J. Schoenwaelder, "Transport Subsystem for the Simple Network Management Protocol (SNMP)", RFC 5590, June 2009.

[RFC5591] Harrington, D. and W. Hardaker, "Transport Security Model for the Simple Network Management Protocol (SNMP)", RFC 5591, June 2009.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

## 12.2. Informative References

[HEARTBEAT] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", Work in Progress, July 2011.

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

[RFC5343] Schoenwaelder, J., "Simple Network Management Protocol (SNMP) Context EngineID Discovery", RFC 5343, September 2008.

[RFC5592] Harrington, D., Salowey, J., and W. Hardaker, "Secure Shell Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5592, June 2009.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC5953] Hardaker, W., "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5953, August 2010.

# Appendix A. Target and Notification Configuration Example

The following sections describe example configuration for the SNMP-TLS-TM-MIB, the SNMP-TARGET-MIB, the NOTIFICATION-MIB, and the SNMP-VIEW-BASED-ACM-MIB.

## A.1. Configuring a Notification Originator

The following row adds the "Joe Cool" user to the "administrators" group:

```
vacmSecurityModel              = 4 (TSM)
vacmSecurityName               = "Joe Cool"
vacmGroupName                  = "administrators"
vacmSecurityToGroupStorageType = 3 (nonVolatile)
vacmSecurityToGroupStatus      = 4 (createAndGo)
```

The following row configures the snmpTlstmAddrTable to use certificate path validation and to require the remote notification receiver to present a certificate for the "server.example.org" identity.

```
snmpTargetAddrName             =  "toNRAddr"
snmpTlstmAddrServerFingerprint =  ""
snmpTlstmAddrServerIdentity    =  "server.example.org"
snmpTlstmAddrStorageType       =  3         (nonVolatile)
snmpTlstmAddrRowStatus         =  4         (createAndGo)
```

The following row configures the snmpTargetAddrTable to send notifications using TLS/TCP to the snmptls-trap port at 192.0.2.1:

```
snmpTargetAddrName              = "toNRAddr"
snmpTargetAddrTDomain           = snmpTLSTCPDomain
snmpTargetAddrTAddress          = "192.0.2.1:10162"
snmpTargetAddrTimeout           = 1500
snmpTargetAddrRetryCount        = 3
snmpTargetAddrTagList           = "toNRTag"
snmpTargetAddrParams            = "toNR"     (MUST match below)
snmpTargetAddrStorageType       = 3          (nonVolatile)
snmpTargetAddrRowStatus         = 4          (createAndGo)
```

The following row configures the snmpTargetParamsTable to send the notifications to "Joe Cool", using authPriv SNMPv3 notifications through the TransportSecurityModel [RFC5591]:

```
snmpTargetParamsName            = "toNR"     (must match above)
snmpTargetParamsMPModel         = 3 (SNMPv3)
snmpTargetParamsSecurityModel   = 4 (TransportSecurityModel)
snmpTargetParamsSecurityName    = "Joe Cool"
snmpTargetParamsSecurityLevel   = 3          (authPriv)
snmpTargetParamsStorageType     = 3          (nonVolatile)
snmpTargetParamsRowStatus       = 4          (createAndGo)
```

## A.2. Configuring TLSTM to Utilize a Simple Derivation of tmSecurityName

The following row configures the snmpTlstmCertToTSNTable to map a validated client certificate, referenced by the client's public X.509 hash fingerprint, to a tmSecurityName using the subjectAltName component of the certificate.

```
snmpTlstmCertToTSNID          = 1
                                (chosen by ordering preference)
snmpTlstmCertToTSNFingerprint = HASH (appropriate fingerprint)
snmpTlstmCertToTSNMapType     = snmpTlstmCertSANAny
snmpTlstmCertToTSNData        = ""  (not used)
snmpTlstmCertToTSNStorageType = 3   (nonVolatile)
snmpTlstmCertToTSNRowStatus   = 4   (createAndGo)
```

This type of configuration should only be used when the naming conventions of the (possibly multiple) Certification Authorities are well understood, so two different principals cannot inadvertently be identified by the same derived tmSecurityName.

## A.3. Configuring TLSTM to Utilize Table-Driven Certificate Mapping

The following row configures the snmpTlstmCertToTSNTable to map a validated client certificate, referenced by the client's public X.509 hash fingerprint, to the directly specified tmSecurityName of "Joe Cool".

```
snmpTlstmCertToTSNID           = 2
                                 (chosen by ordering preference)
snmpTlstmCertToTSNFingerprint  = HASH (appropriate fingerprint)
snmpTlstmCertToTSNMapType      = snmpTlstmCertSpecified
snmpTlstmCertToTSNSecurityName = "Joe Cool"
snmpTlstmCertToTSNStorageType  = 3  (nonVolatile)
snmpTlstmCertToTSNRowStatus    = 4  (createAndGo)
```

Author's Address

Wes Hardaker
SPARTA, Inc.
P.O. Box 382
Davis, CA 95617
USA

Phone: +1 530 792 1913
EMail: ietf@hardakers.net