

ISMS  
Internet-Draft  
Intended status: Standards Track  
Expires: August 6, 2010

W. Hardaker  
Sparta, Inc.  
February 2, 2010

**Transport Layer Security (TLS) Transport Model for SNMP  
draft-ietf-isms-dtls-tm-08.txt**

**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 a 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 sending of SNMP messages over TLS/TCP, DTLS/UDP and DTLS/SCTP. 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 or SCTP) 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 Internet-Draft is submitted to IETF in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at

<http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on August 6, 2010.

#### Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.



## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction . . . . .</a>	<a href="#">5</a>
<a href="#">1.1.</a>	<a href="#">Conventions . . . . .</a>	<a href="#">7</a>
<a href="#">2.</a>	<a href="#">The Transport Layer Security Protocol . . . . .</a>	<a href="#">8</a>
<a href="#">3.</a>	<a href="#">How the TLSTM fits into the Transport Subsystem . . . . .</a>	<a href="#">8</a>
<a href="#">3.1.</a>	<a href="#">Security Capabilities of this Model . . . . .</a>	<a href="#">10</a>
<a href="#">3.1.1.</a>	<a href="#">Threats . . . . .</a>	<a href="#">10</a>
<a href="#">3.1.2.</a>	<a href="#">Message Protection . . . . .</a>	<a href="#">12</a>
<a href="#">3.1.3.</a>	<a href="#">(D)TLS Sessions . . . . .</a>	<a href="#">12</a>
<a href="#">3.2.</a>	<a href="#">Security Parameter Passing . . . . .</a>	<a href="#">13</a>
<a href="#">3.3.</a>	<a href="#">Notifications and Proxy . . . . .</a>	<a href="#">14</a>
<a href="#">4.</a>	<a href="#">Elements of the Model . . . . .</a>	<a href="#">14</a>
<a href="#">4.1.</a>	<a href="#">X.509 Certificates . . . . .</a>	<a href="#">15</a>
<a href="#">4.1.1.</a>	<a href="#">Provisioning for the Certificate . . . . .</a>	<a href="#">15</a>
<a href="#">4.2.</a>	<a href="#">Messages . . . . .</a>	<a href="#">16</a>
<a href="#">4.3.</a>	<a href="#">SNMP Services . . . . .</a>	<a href="#">16</a>
<a href="#">4.3.1.</a>	<a href="#">SNMP Services for an Outgoing Message . . . . .</a>	<a href="#">17</a>
<a href="#">4.3.2.</a>	<a href="#">SNMP Services for an Incoming Message . . . . .</a>	<a href="#">17</a>
<a href="#">4.4.</a>	<a href="#">Cached Information and References . . . . .</a>	<a href="#">18</a>
<a href="#">4.4.1.</a>	<a href="#">TLS Transport Model Cached Information . . . . .</a>	<a href="#">18</a>
<a href="#">4.4.1.1.</a>	<a href="#">tmSecurityName . . . . .</a>	<a href="#">19</a>
<a href="#">4.4.1.2.</a>	<a href="#">tmSessionID . . . . .</a>	<a href="#">19</a>
<a href="#">4.4.1.3.</a>	<a href="#">Session State . . . . .</a>	<a href="#">19</a>
<a href="#">5.</a>	<a href="#">Elements of Procedure . . . . .</a>	<a href="#">19</a>
<a href="#">5.1.</a>	<a href="#">Procedures for an Incoming Message . . . . .</a>	<a href="#">20</a>
<a href="#">5.1.1.</a>	<a href="#">DTLS Processing for Incoming Messages . . . . .</a>	<a href="#">20</a>
<a href="#">5.1.2.</a>	<a href="#">Transport Processing for Incoming SNMP Messages . . . . .</a>	<a href="#">22</a>
<a href="#">5.2.</a>	<a href="#">Procedures for an Outgoing SNMP Message . . . . .</a>	<a href="#">23</a>
<a href="#">5.3.</a>	<a href="#">Establishing or Accepting a Session . . . . .</a>	<a href="#">25</a>
<a href="#">5.3.1.</a>	<a href="#">Establishing a Session as a Client . . . . .</a>	<a href="#">25</a>
<a href="#">5.3.2.</a>	<a href="#">Accepting a Session as a Server . . . . .</a>	<a href="#">27</a>
<a href="#">5.4.</a>	<a href="#">Closing a Session . . . . .</a>	<a href="#">28</a>
<a href="#">6.</a>	<a href="#">MIB Module Overview . . . . .</a>	<a href="#">28</a>
<a href="#">6.1.</a>	<a href="#">Structure of the MIB Module . . . . .</a>	<a href="#">28</a>
<a href="#">6.2.</a>	<a href="#">Textual Conventions . . . . .</a>	<a href="#">29</a>
<a href="#">6.3.</a>	<a href="#">Statistical Counters . . . . .</a>	<a href="#">29</a>
<a href="#">6.4.</a>	<a href="#">Configuration Tables . . . . .</a>	<a href="#">29</a>
<a href="#">6.4.1.</a>	<a href="#">Notifications . . . . .</a>	<a href="#">29</a>
<a href="#">6.5.</a>	<a href="#">Relationship to Other MIB Modules . . . . .</a>	<a href="#">29</a>
<a href="#">6.5.1.</a>	<a href="#">MIB Modules Required for IMPORTS . . . . .</a>	<a href="#">30</a>
<a href="#">7.</a>	<a href="#">MIB Module Definition . . . . .</a>	<a href="#">30</a>
<a href="#">8.</a>	<a href="#">Operational Considerations . . . . .</a>	<a href="#">51</a>
<a href="#">8.1.</a>	<a href="#">Sessions . . . . .</a>	<a href="#">52</a>
<a href="#">8.2.</a>	<a href="#">Notification Receiver Credential Selection . . . . .</a>	<a href="#">52</a>
<a href="#">8.3.</a>	<a href="#">contextEngineID Discovery . . . . .</a>	<a href="#">53</a>
<a href="#">8.4.</a>	<a href="#">Transport Considerations . . . . .</a>	<a href="#">53</a>
<a href="#">9.</a>	<a href="#">Security Considerations . . . . .</a>	<a href="#">53</a>



<a href="#">9.1.</a>	<a href="#">Certificates, Authentication, and Authorization . . . . .</a>	<a href="#">53</a>
<a href="#">9.2.</a>	<a href="#">Use with SNMPv1/SNMPv2c Messages . . . . .</a>	<a href="#">54</a>
<a href="#">9.3.</a>	<a href="#">MIB Module Security . . . . .</a>	<a href="#">55</a>
<a href="#">10.</a>	<a href="#">IANA Considerations . . . . .</a>	<a href="#">56</a>
<a href="#">11.</a>	<a href="#">Acknowledgements . . . . .</a>	<a href="#">57</a>
<a href="#">12.</a>	<a href="#">References . . . . .</a>	<a href="#">58</a>
<a href="#">12.1.</a>	<a href="#">Normative References . . . . .</a>	<a href="#">58</a>
<a href="#">12.2.</a>	<a href="#">Informative References . . . . .</a>	<a href="#">59</a>
<a href="#">Appendix A.</a>	<a href="#">(D)TLS Overview . . . . .</a>	<a href="#">60</a>
<a href="#">A.1.</a>	<a href="#">The (D)TLS Record Protocol . . . . .</a>	<a href="#">60</a>
<a href="#">A.2.</a>	<a href="#">The (D)TLS Handshake Protocol . . . . .</a>	<a href="#">61</a>
<a href="#">Appendix B.</a>	<a href="#">PKIX Certificate Infrastructure . . . . .</a>	<a href="#">62</a>
<a href="#">Appendix C.</a>	<a href="#">Target and Notification Configuration Example . . . . .</a>	<a href="#">63</a>
<a href="#">C.1.</a>	<a href="#">Configuring the Notification Originator . . . . .</a>	<a href="#">64</a>
<a href="#">C.2.</a>	<a href="#">Configuring the Command Responder . . . . .</a>	<a href="#">64</a>
	<a href="#">Author's Address . . . . .</a>	<a href="#">65</a>



## 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 key 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 or SCTP) 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 key infrastructures. This document supports sending of SNMP messages over TLS/TCP, DTLS/UDP and DTLS/SCTP.

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.

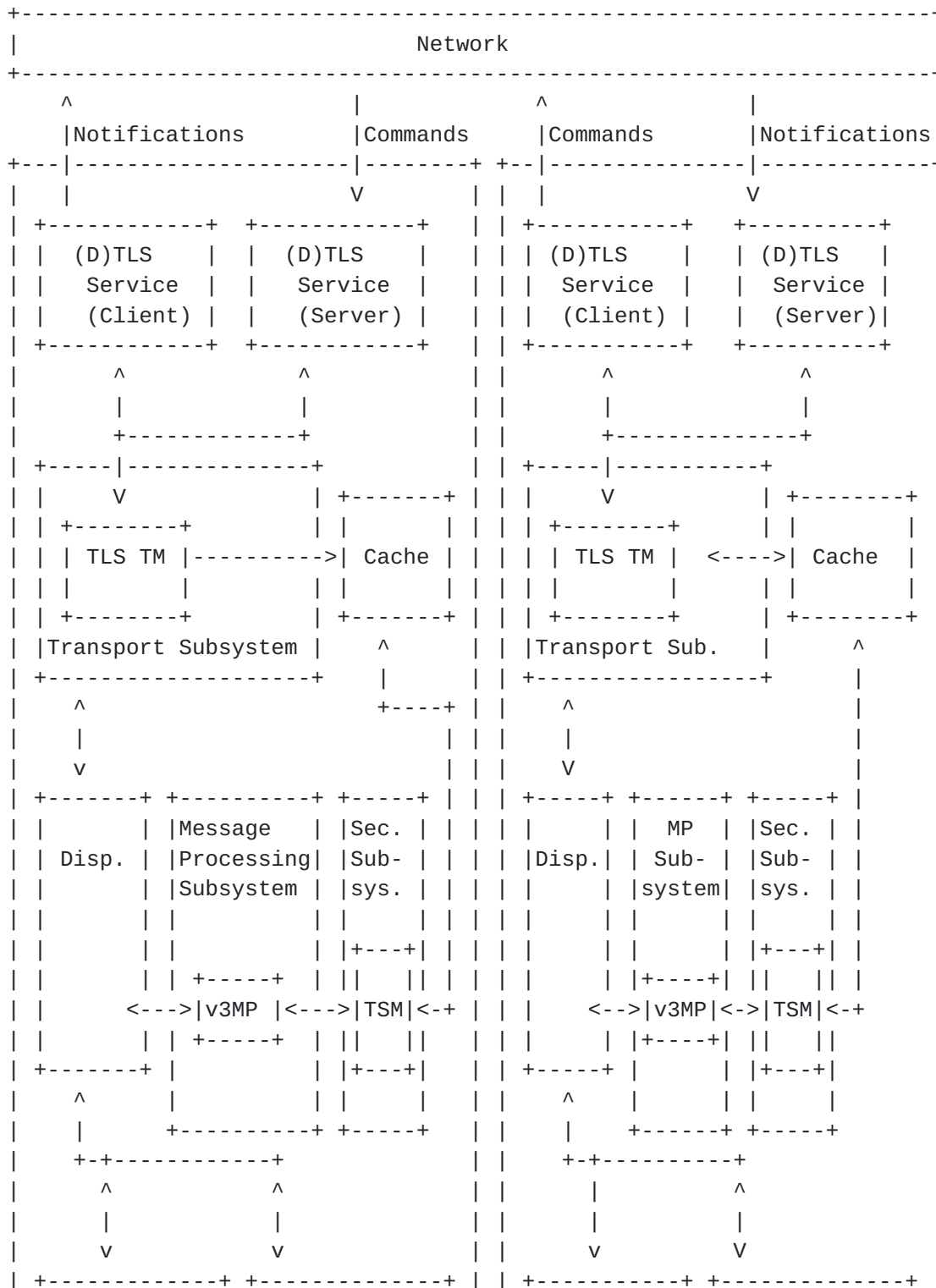
Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (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. One entity contains a command responder and notification originator application, and the other a command generator and notification responder 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 which is defined in [\[RFC5591\]](#).





		COMMAND			NOTIFICATION				COMMAND			NOTIFICATION		
		RESPONDER			ORIGINATOR				GENERATOR			RECEIVER		
		application			application				application			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 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 standard defined 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 [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, 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, this document (unless otherwise specified) is referring to 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 [\[RFC2119\]](#).

## **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. An overview of (D)TLS can be found in section [Appendix A](#). 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 and the Transport Subsystem.

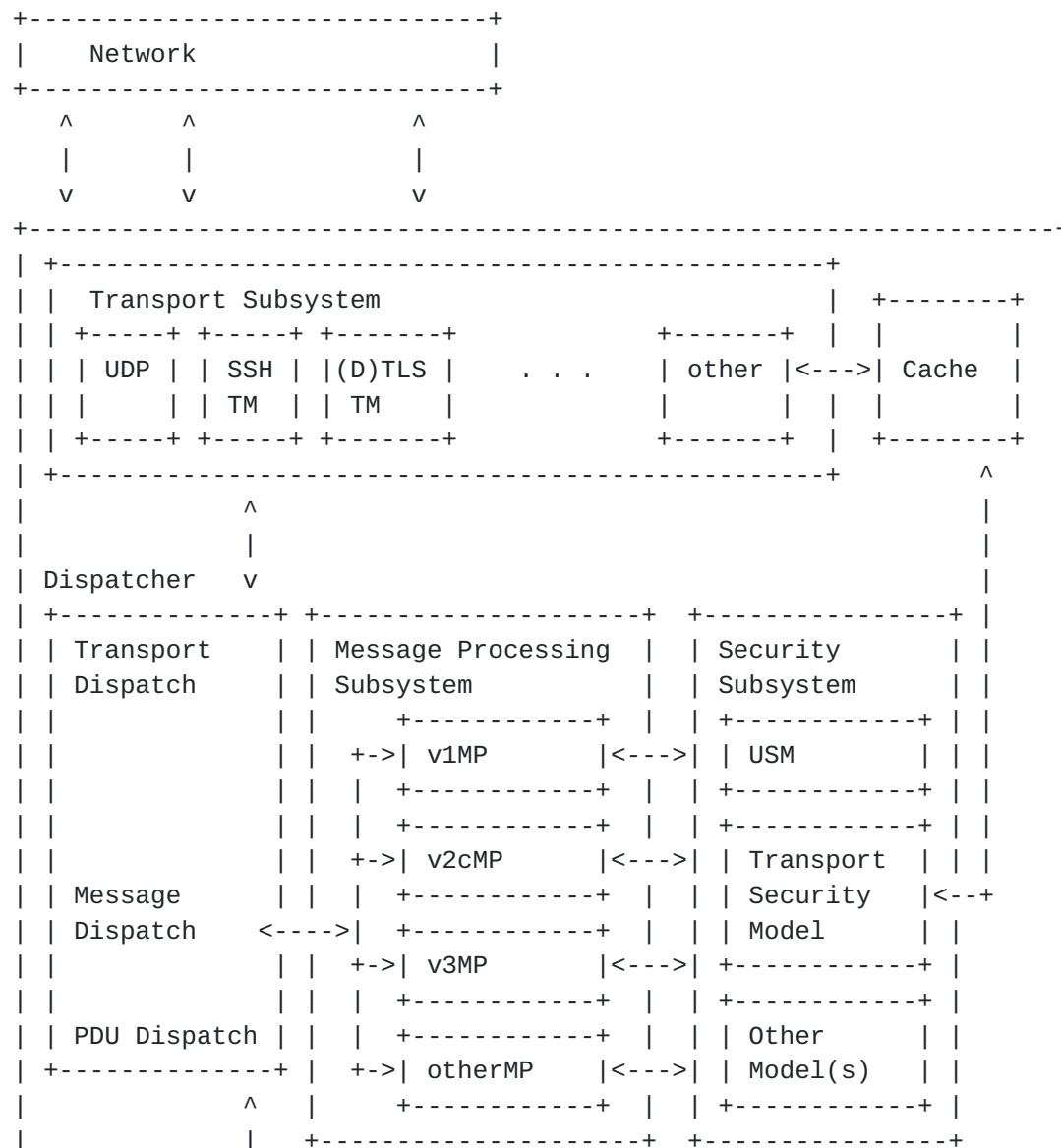
The TLS Transport Model will establish a session between itself and the TLS Transport Model of another SNMP engine. The sending transport model passes unencrypted and unauthenticated messages from the Dispatcher to (D)TLS to be encrypted and authenticated, and the receiving transport model accepts decrypted 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 a (D)TLS 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 fits into the architecture described in [RFC3411](#) and the Transport Subsystem:







The TLSTM verifies the identity of the (D)TLS server through the use of the (D)TLS protocol and X.509 certificates. The TLS Transport Model MUST support 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 which 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 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. The TLS Transport Model SHOULD support the 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, datagram-based security protocols like DTLS are susceptible to a variety of denial of service attacks because they are more vulnerable to spoofed messages.

In order to counter these attacks, DTLS borrows the stateless cookie technique used by Photuris [\[RFC2522\]](#) and IKEv2 [\[RFC4306\]](#) and is described fully in [section 4.2.1 of \[RFC4347\]](#). This mechanism, though, does not provide any defense against denial of service attacks mounted from valid IP addresses. DTLS Transport Model server implementations MUST support DTLS cookies.

Implementations are not required to perform the stateless cookie exchange for every DTLS handshake, but in environments where an overload on server side resources is detectable by the implementation it is RECOMMENDED that the cookie exchange is utilized by the implementation.

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:

- o without authentication and without privacy (noAuthNoPriv)
- o with authentication but without privacy (authNoPriv)
- o 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 session 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 Sessions**

(D)TLS sessions 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 session if needed, the (D)TLS session will already exist for an incoming message.



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

DTLS sessions, 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 and DTLS over SCTP sessions, on the other hand, do not require a unique pairing of address and port attributes since their lower layer protocols (TCP and SCTP) already provide adequate session framing. But they must still provide a unique `tlstmSessionID` for referencing the session.

As an implementation hint: although the `tlstmSessionID` may be the same as the (D)TLS internal `SessionID` caution must be exercised since the (D)TLS internal `SessionID` may change over the life of the connection as seen by the TLSTM (for example during renegotiation). The `tlstmSessionID` identifier MUST NOT change during the entire duration of the session from the TLSTM's perspective even if the TLS internal session identifier does change.

### **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 address and port) 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 sessions 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 is 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, snmpDTLSUDPDDomain, or snmpDTLSSCTPDomain 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 tlstmParamsClientFingerprint parameter of the tlstmParamsTable should be set a value that refers to a locally held certificate 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. A brief overview of X.509 certificate infrastructure can be found in [Appendix B](#).

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. 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 are provisioned with certificates, which are signed by trusted certificate authorities (possibly the certificate itself). Furthermore, SNMP entities will most commonly need to be provisioned with root certificates which represent the list of trusted certificate authorities that an SNMP entity can use for certificate verification. SNMP entities SHOULD also be provisioned with a 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 tlstmCertToTSNTable. This table allows mapping of incoming connections to tmSecurityNames through defined transformations. The transformations defined in the TLSTM-MIB include:

- o Mapping a certificate's subjectAltName or CommonName components to a tmSecurityName, or
- o 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 tlstmCertToTSNTable mapping exists before performing certificate verification to avoid expending computational resources associated with certificate verification.

Enterprise configurations are encouraged to map a "subjectAltName" component of the X.509 certificate to the TLSTM specific tmSecurityName. The authenticated identity can be obtained by the

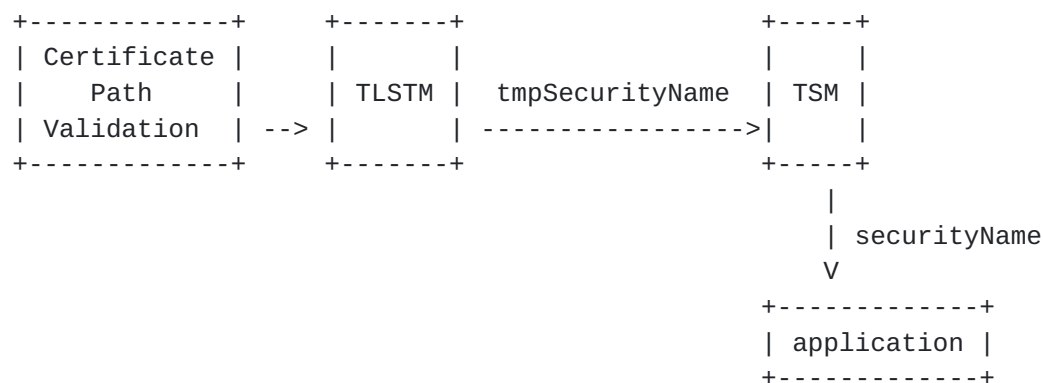


TLS Transport Model by extracting the subjectAltName(s) from the peer's certificate. The receiving application will then have an appropriate tmSecurityName for use by other SNMPv3 components like an access control model.

An example of this type of mapping setup can be found in [Appendix C](#).

This tmSecurityName may be later translated from a TLSTM specific tmSecurityName to a 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.

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



## 4.2. Messages

As stated in [Section 4.1.1 of \[RFC4347\]](#), each DTLS record must fit within a single DTLS datagram. The TLSTM SHOULD prohibit SNMP messages from being sent that exceeds the maximum DTLS message size. The TLSTM implementation SHOULD return an error when the DTLS message size would be exceeded and the message won't be sent.

## 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 snmpTLSDomain, the snmpDTLSUDPDDomain and the snmpDTLSSCTPDdomain 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 field.

**tmStateReference:** A reference to tmState to be used when securing outgoing messages.

#### **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 snmpTLSDomain, the snmpDTLSUDPDDomain and the snmpDTLSSCTPDDomain 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 and the (D)TLS transport layer data has been removed.

**incomingMessageLength:** The length of the incomingMessage field.

**tmStateReference:** A reference to tmSecurityData to be used by the security model.

#### **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" [[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 an TLSTM session. Messages MUST NOT be sent through an existing (D)TLS session 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 TLSTM-MIB's tlstmCertToTSNTable DESCRIPTION clause.

On the (D)TLS client side of a connection the tmSecurityName is presented to the TLS Transport Model by the application (possibly because of configuration specified in the SNMP-TARGET-MIB).

The securityName MAY be derived from the tmSecurityName by a Security Model and MAY be used to configure notifications and access controls in MIB modules. Transport Models SHOULD generate a predictable tmSecurityName so operators will know what to use when configuring MIB modules that use securityNames derived from tmSecurityNames.

#### **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 session available for sending a corresponding outgoing message is the same (D)TLS session 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 sessions, which is specifically needed for DTLS over UDP sessions. It is assumed that TLS and DTLS/SCP 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 sessions.

### **5.1.1. DTLS Processing for Incoming Messages**

DTLS over UDP is significantly different in terms of session handling than when TLS or DTLS is run over session based streaming protocols like TCP or SCTP. Specifically, the DTLS protocol, when run over UDP, does not have a session identifier that allows implementations to determine through which session a packet arrived. It is critical, however, that implementations are always able to derive a tlstmSessionID from any session demultiplexing process. 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 easily disambiguate between multiple sessions from a host to the same port on a server.

A process for demultiplexing multiple DTLS sessions arriving over UDP must be incorporated into the procedures for processing an incoming



message. The steps in this section describe one possible method to accomplish this, although any implementation-dependent method should be suitable as long as the results are deterministic. The important output results from the steps in this process are the `transportDomain`, the `transportAddress`, the `wholeMessage`, the `wholeMessageLength`, and a unique implementation-dependent session identifier (`tlstmSessionID`).

This demultiplexing procedure assumes that upon session establishment an entry in a local transport mapping table is created in the Transport Model's Local Configuration Datastore (LCD). The transport mapping table's entry should map a unique combination of the remote address, remote port number, local address and local port number to an implementation-dependent `tlstmSessionID`.

- 1) The TLS Transport Model examines the raw UDP message, in an implementation-dependent manner.
- 2) The TLS Transport Model queries the LCD using the transport parameters (source and destination addresses and ports) to determine if a session already exists.

If a matching entry in the LCD does not exist then the message is passed to DTLS for processing without a corresponding `tlstmSessionID`. The incoming packet may result in a new session being established if the receiving entity is acting as a DTLS server. If DTLS returns success then stop processing of this message. If DTLS returns an error then increment the `snmpTlstmSessionNoSessions` counter and stop processing the message.

Note that an entry would already exist if the client and server's session establishment procedures had been successfully completed previously (as described both above and in [Section 5.3](#)) even if no message had yet been sent through the newly established session. An entry may not exist, however, if a message not intended the SNMP entity was routed to it by mistake. An entry might also be missing because of a "broken" session (see operational considerations).

- 3) Retrieve the `tlstmSessionID` from the LCD.
- 4) The UDP packet and the `tlstmSessionID` are passed to DTLS for integrity checking and decryption. If processing does not return an `incomingMessage` and an `incomingMessageLength` then processing stops.





- 5) 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`, `snmpDTLSUDPDomain` or `snmpDTLSSCTPDomain` as appropriate.



tmTransportAddress = The address the message originated from.

tmSecurityLevel = The derived tmSecurityLevel for the session,  
as discussed in [Section 3.1.2](#) and [Section 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 (D)TLS 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, snmpDTLSUDPDDomain,  
                                -- or snmpDTLSSCTPDDomain  
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 either `tmSessionID` is 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 a `snmpTlstmSessionNoSessions` error to the calling module and stop 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 not a 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 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 session 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 session:

```
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 a SNMP over (D)TLS session 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 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 a SNMP-TARGET-MIB based operation, the certificate will potentially have been identified via the `tlstmParamsTable` mapping and the `cipher_suites` will have to be taken from system-wide or implementation-specific configuration. 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





session. 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 a `tlstmServerCertificateUnknown` or `tlstmServerInvalidCertificate` notification SHOULD be sent as appropriate. Reasons for server certificate invalidation includes, but is 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 and the client certificate has been transmitted.

If the connection is being established from configuration based on SNMP-TARGET-MIB configuration then the procedures in the `tlstmAddrTable` DESCRIPTION clause should be followed to determine if the presented identity matches the expectations of the configuration. Validation procedures (like the path validation procedures defined in [\[RFC5280\]](#) or through the use of fingerprints as defined by the `tlstmAddrServerIdentity` column) MUST be followed. If a server identity name has been configured in the `tlstmAddrServerIdentity` column then this reference identity must be compared against the presented identity (for example using procedures described in [\[I-D.saintandre-tls-server-id-check\]](#)).

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.



- 5) (D)TLS provides assurance that the authenticated identity has been signed by a trusted configured certificate 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, the `snmpTlstmSessionInvalidServerCertificates` object is incremented. If the session can not be opened for any reason at all, including cryptographic verification 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 session 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 that it is able to verify the client's credentials for. 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 `tlstmCertToTSNTable`. 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 `tlstmCertToTSNTable` 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 session 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 `tlstmCertToTSNTable` 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 `tlstmCertToTSNTable`) then the session establishment MUST fail, the `snmpTlstmSessionInvalidClientCertificates` object is incremented. If the session can not 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, UDP or SCTP 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 session. This SHOULD 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 C](#).

#### **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:

- o A new TransportAddress format for describing (D)TLS connection addressing requirements.
- o 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 TLSTM-MIB defines some counters that can provide network managers with information about (D)TLS session usage and potential errors that a MIB-instrumented device may be experiencing.

## **6.4. Configuration Tables**

The TLSTM-MIB defines configuration tables that a manager can use for configuring a MIB-instrumented 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 securityNames.

### **6.4.1. Notifications**

The TLSTM-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 (tlstmServerCertificateUnknown) or because cryptographic validation failed (tlstmServerInvalidCertificate).

## **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 TLSTM-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 TLSTM-MIB module contained in this document is for managing TLS Transport Model information.

#### **6.5.1. MIB Modules Required for IMPORTS**

The TLSTM-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**

TLSTM-MIB DEFINITIONS ::= BEGIN

#### **IMPORTS**

```
MODULE-IDENTITY, OBJECT-TYPE,
OBJECT-IDENTITY, snmpModules, snmpDomains,
Counter32, Unsigned32, NOTIFICATION-TYPE
    FROM SNMPv2-SMI
TEXTUAL-CONVENTION, TimeStamp, RowStatus, StorageType,
AutonomousType
    FROM SNMPv2-TC
MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP
    FROM SNMPv2-CONF
SnmpAdminString
    FROM SNMP-FRAMEWORK-MIB
snmpTargetParamsName, snmpTargetAddrName
    FROM SNMP-TARGET-MIB
;
```

#### **tlstmMIB MODULE-IDENTITY**

```
LAST-UPDATED "201002020000Z"
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

## Co-editors:

Wes Hardaker  
Sparta, Inc.  
P.O. Box 382  
Davis, CA 95617  
USA  
ietf@hardakers.net

"

## DESCRIPTION "

The TLS Transport Model MIB

Copyright (c) 2010 IETF Trust and the persons identified as  
the document authors. 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>).

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

-- NOTE to RFC editor: replace XXXX with actual RFC number  
-- for this document and remove this note

REVISION "201002020000Z"

DESCRIPTION "The initial version, published in RFC XXXX."

-- NOTE to RFC editor: replace XXXX with actual RFC number  
-- for this document and remove this note

::= { snmpModules xxxx }

-- RFC Ed.: replace xxxx with IANA-assigned number and  
-- remove this note

-- \*\*\*\*\*  
-- subtrees of the TLSTM-MIB  
-- \*\*\*\*\*

tlstmNotifications OBJECT IDENTIFIER ::= { tlstmMIB 0 }  
tlstmIdentities OBJECT IDENTIFIER ::= { tlstmMIB 1 }  
tlstmObjects OBJECT IDENTIFIER ::= { tlstmMIB 2 }



```
tlstmConformance OBJECT IDENTIFIER ::= { tlstmMIB 3 }
```

```
-- *****  
-- tlstmObjects - Objects  
-- *****
```

```
snmpTLSTCPDomain OBJECT-IDENTITY
```

```
STATUS current
```

```
DESCRIPTION
```

```
"The SNMP over TLS 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."
```

```
::= { snmpDomains xx }
```

```
-- RFC Ed.: replace xx with IANA-assigned number and  
-- remove this note
```

```
-- RFC Ed.: replace 'tls' with the actual IANA assigned prefix string  
-- if 'tls' is not assigned to this document.
```

```
snmpDTLSUDPDDomain OBJECT-IDENTITY
```

```
STATUS current
```

```
DESCRIPTION
```

```
"The SNMP over DTLS/UDP transport domain. The corresponding  
transport address is of type SnmpTLSAddress.
```

```
The securityName prefix to be associated with the  
snmpDTLSUDPDDomain is 'dudp'. This prefix may be used by  
security models or other components to identify which secure  
transport infrastructure authenticated a securityName."
```

```
::= { snmpDomains yy }
```

```
-- RFC Ed.: replace yy with IANA-assigned number and  
-- remove this note
```

```
-- RFC Ed.: replace 'dudp' with the actual IANA assigned prefix string  
-- if 'dudp' is not assigned to this document.
```

```
snmpDTLSSCTPDDomain OBJECT-IDENTITY
```

```
STATUS current
```



## DESCRIPTION

"The SNMP over DTLS/SCTP transport domain. The corresponding transport address is of type SnmpTLSAddress.

The securityName prefix to be associated with the snmpDTLSSCTPDomain is 'dsct'. This prefix may be used by security models or other components to identify which secure transport infrastructure authenticated a securityName."

::= { snmpDomains zz }

-- RFC Ed.: replace zz with IANA-assigned number and  
-- remove this note

-- RFC Ed.: replace 'dsct' with the actual IANA assigned prefix string  
-- if 'dsct' is not assigned to this document.

SnmpTLSAddress ::= TEXTUAL-CONVENTION

DISPLAY-HINT "1a"

STATUS current

## DESCRIPTION

"Represents a IPv4 address, an IPv6 address or an 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, 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 [RFC1033](#)); internationalized hostnames are encoded in US-ASCII as specified in [RFC 3490](#). 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 1033](#): DOMAIN ADMINISTRATORS OPERATIONS GUIDE  
[RFC 3490](#): Internationalizing Domain Names in Applications  
[RFC 3986](#): Uniform Resource Identifier (URI): Generic Syntax  
[RFC 5246](#): The Transport Layer Security (TLS) Protocol Version 1.2  
"

SYNTAX           OCTET STRING (SIZE (1..255))

Fingerprint ::= TEXTUAL-CONVENTION

DISPLAY-HINT "1x:254x"

STATUS           current

#### DESCRIPTION

"A Fingerprint value that can be used to uniquely reference other data of potentially arbitrary length.

A Fingerprint 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 ([RFC5246](#)). The remaining octets are filled using the results of the hashing algorithm.

This TEXTUAL-CONVENTION allows for a zero-length (blank) Fingerprint 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 tlstmCertToTSNTable

tlstmCertToTSNMIdentities OBJECT IDENTIFIER ::= { tlstmIdentities 1 }

tlstmCertSpecified 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 tlstmCertToTSNData column. The tlstmCertToTSNData column must contain a non-zero length SnmpAdminString compliant value or the mapping described in this row must be considered a failure."

::= { tlstmCertToTSNMIdentities 1 }

tlstmCertSANRFC822Name 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 lower case.

Example rfc822Name Field: FooBar@Example.COM  
is mapped to tmSecurityName: FooBar@example.com"

::= { tlstmCertToTSNMIdentities 2 }

tlstmCertSANDNSName OBJECT-IDENTITY

STATUS current

DESCRIPTION "Maps a subjectAltName's dNSName to a tmSecurityName by directly passing the value without any transformations."

::= { tlstmCertToTSNMIdentities 3 }

tlstmCertSANIpAddress 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 hexadecimal string without any colon



separators.

Note that the resulting length is the maximum length supported by the View-Based Access Control Model (VACM). Note that 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."

```
::= { tlstmCertToTSNMIdentities 4 }
```

`tlstmCertSANAny` OBJECT-IDENTITY

STATUS current

DESCRIPTION "Maps any of the following fields using the corresponding mapping algorithms:

-----+-----	
Type	Algorithm
-----+-----	
rfc822Name	tlstmCertSANRFC822Name
dNSName	tlstmCertSANDNSName
iPAddress	tlstmCertSANIpAddress
-----+-----	

The first matching `subjectAltName` value found in the certificate of the above types MUST be used when deriving the `tmSecurityName`."

```
::= { tlstmCertToTSNMIdentities 5 }
```

`tlstmCertCommonName` OBJECT-IDENTITY

STATUS current

DESCRIPTION "Maps a certificate's `CommonName` to a `tmSecurityName` by directly passing the value without any transformations."

```
::= { tlstmCertToTSNMIdentities 6 }
```

-- The `snmpTlstmSession` Group

```
snmpTlstmSession          OBJECT IDENTIFIER ::= { tlstmObjects 1 }
```

`snmpTlstmSessionOpens` OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of times an `openSession()` request has been executed as an (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 an (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 server has accepted a (D)TLS session and at least one SNMP message has been accepted 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 an (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 an (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 tlstmCertToTSNTable."

::= { 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 an (D)TLS client because the server certificate presented by a SNMP over (D)TLS server was invalid because no configured fingerprint or CA was acceptable to validate it. This may result because there was no entry in the tlstmAddrTable or because no path could be found to a known certificate authority."

::= { 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 an (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. I.E., 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

tlstmConfig OBJECT IDENTIFIER ::= { tlstmObjects 2 }

-- Certificate mapping

tlstmCertificateMapping OBJECT IDENTIFIER ::= { tlstmConfig 1 }

tlstmCertToTSNCount OBJECT-TYPE

SYNTAX Unsigned32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"A count of the number of entries in the tlstmCertToTSNTable"

::= { tlstmCertificateMapping 1 }

tlstmCertToTSNTableLastChanged OBJECT-TYPE

SYNTAX TimeStamp

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The value of sysUpTime.0 when the tlstmCertToTSNTable  
was last modified through any means, or 0 if it has not been  
modified since the command responder was started."

::= { tlstmCertificateMapping 2 }

tlstmCertToTSNTable OBJECT-TYPE

SYNTAX SEQUENCE OF TlstmCertToTSNEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A table listing the fingerprints of X.509 certificates known  
to the entity and the associated method for determining the  
SNMPv3 security name from a certificate.

On an incoming (D)TLS/SNMP connection the client's presented  
certificate must be examined and validated based on an  
established trusted path from a CA certificate or self-signed  
public certificate (e.g. [RFC5280](#)). This table provides a  
mapping from a validated certificate to a tmSecurityName.  
This table does not provide any mechanisms for uploading  
trusted certificates; the transfer of any needed trusted



certificates for path validation is expected to occur through an out-of-band transfer.

Once the authenticity of a certificate has been verified, 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 `tlstmCertToTSNID` value. Each row's `tlstmCertToTSNFingerprint` value determines whether the row is a match for the incoming connection:

- 1) If the row's `tlstmCertToTSNFingerprint` value identifies the presented certificate then consider the row as a successful match.
- 2) If the row's `tlstmCertToTSNFingerprint` 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 `tlstmCertToTSNMapType` value can be used to determine how the `tmSecurityName` to associate with the session should be determined. See the `tlstmCertToTSNMapType` 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.

Missing values of `tlstmCertToTSNID` are acceptable and implementations should continue to the next highest numbered row. E.G., the table may legally contain only two rows with `tlstmCertToTSNID` values of 10 and 20.

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."

```
::= { tlstmCertificateMapping 3 }
```

tlstmCertToTSNEntry OBJECT-TYPE

SYNTAX TlstmCertToTSNEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A row in the tlstmCertToTSNTable that specifies a mapping for an incoming (D)TLS certificate to a tmSecurityName to use for a connection."

INDEX { tlstmCertToTSNID }

```
::= { tlstmCertToTSNTable 1 }
```

TlstmCertToTSNEntry ::= SEQUENCE {

tlstmCertToTSNID Unsigned32,  
tlstmCertToTSNFingerprint Fingerprint,  
tlstmCertToTSNMapType AutonomousType,  
tlstmCertToTSNData OCTET STRING,  
tlstmCertToTSNStorageType StorageType,  
tlstmCertToTSNRowStatus RowStatus

}

tlstmCertToTSNID 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."

```
::= { tlstmCertToTSNEntry 1 }
```

tlstmCertToTSNFingerprint OBJECT-TYPE

SYNTAX Fingerprint (SIZE(1..255))

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"A cryptographic hash of a 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 `tlstmCertToTSNMapType` column."  
 ::= { `tlstmCertToTSNEntry` 2 }

`tlstmCertToTSNMapType` 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 `tlstmCertToTSNFingerprint` matches to look for a mapping that succeeds."

DEFVAL { `tlstmCertSpecified` }

::= { `tlstmCertToTSNEntry` 3 }

`tlstmCertToTSNData` 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 `tlstmCertToTSNMapType` 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 { "" }

::= { `tlstmCertToTSNEntry` 4 }

`tlstmCertToTSNStorageType` 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` }

::= { `tlstmCertToTSNEntry` 5 }



**tlstmCertToTSNRowStatus 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, a manager 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 tlstmParamsRowStatus column is 'notReady'.

In particular, a newly created row cannot be made active until the corresponding tlstmCertToTSNFingerprint, tlstmCertToTSNMapType, and tlstmCertToTSNData columns have been set.

The following objects may not be modified while the value of this object is active(1):

- tlstmCertToTSNFingerprint
- tlstmCertToTSNMapType
- tlstmCertToTSNData

An attempt to set these objects while the value of tlstmParamsRowStatus is active(1) will result in an inconsistentValue error."

::= { tlstmCertToTSNEntry 6 }

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

**tlstmParamsCount OBJECT-TYPE**

SYNTAX        Unsigned32  
MAX-ACCESS   read-only  
STATUS        current  
DESCRIPTION

"A count of the number of entries in the tlstmParamsTable"

::= { tlstmCertificateMapping 4 }

**tlstmParamsTableLastChanged OBJECT-TYPE**

SYNTAX        TimeStamp  
MAX-ACCESS   read-only  
STATUS        current  
DESCRIPTION

"The value of sysUpTime.0 when the tlstmParamsTable was last modified through any means, or 0 if it has not been modified since the command responder was started."



```
::= { tlstmCertificateMapping 5 }
```

tlstmParamsTable OBJECT-TYPE

SYNTAX SEQUENCE OF TlstmParamsEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"This table is used by a (D)TLS client when a (D)TLS session 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."

```
::= { tlstmCertificateMapping 6 }
```

tlstmParamsEntry OBJECT-TYPE

SYNTAX TlstmParamsEntry

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 }

```
::= { tlstmParamsTable 1 }
```

```
TlstmParamsEntry ::= SEQUENCE {
```

```
    tlstmParamsClientFingerprint Fingerprint,
```

```
    tlstmParamsStorageType StorageType,
```

```
    tlstmParamsRowStatus RowStatus
```

```
}
```

tlstmParamsClientFingerprint OBJECT-TYPE

SYNTAX Fingerprint

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"A cryptographic hash of a X.509 certificate. This object should store the hash of a locally held X.509 certificate that should be used when initiating a (D)TLS connection as a (D)TLS client."

```
::= { tlstmParamsEntry 1 }
```



**tlstmParamsStorageType 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 }

::= { tlstmParamsEntry 2 }

**tlstmParamsRowStatus 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, a manager 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 tlstmParamsRowStatus column is 'notReady'.

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

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

An attempt to set these objects while the value of tlstmParamsRowStatus is active(1) will result in an inconsistentValue error."

::= { tlstmParamsEntry 3 }

**tlstmAddrCount OBJECT-TYPE**

SYNTAX Unsigned32

MAX-ACCESS read-only

STATUS current

## DESCRIPTION

"A count of the number of entries in the tlstmAddrTable"

::= { tlstmCertificateMapping 7 }

**tlstmAddrTableLastChanged OBJECT-TYPE**





SYNTAX       TimeStamp

MAX-ACCESS   read-only

STATUS       current

DESCRIPTION

"The value of sysUpTime.0 when the tlstmAddrTable was last modified through any means, or 0 if it has not been modified since the command responder was started."

::= { tlstmCertificateMapping 8 }

tlstmAddrTable OBJECT-TYPE

SYNTAX       SEQUENCE OF TlstmAddrEntry

MAX-ACCESS   not-accessible

STATUS       current

DESCRIPTION

"This table is used by a (D)TLS client when a (D)TLS session 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 validate the certificate that the server presents.

If there is a row in this table corresponding to the entry in the SNMP-TARGET-MIB that was used to establish the session (and that row is active), then the fingerprint of the server's presented certificate is compared with the value of the tlstmAddrServerFingerprint column. If fingerprint does not match, then the connection MUST NOT be established.

If the row exists with a zero-length tlstmAddrServerFingerprint value and the certificate can be validated through another certificate validation path (such as the path validation procedures defined in [[RFC5280](#)]) then the server's presented identity should be checked against the value of the tlstmAddrServerIdentity column. If the server's identity does not match the reference identity found in the tlstmAddrServerIdentity column then the connection MUST NOT be established.

A tlstmAddrServerIdentity may contain a single ASCII '\*' character (ASCII code 0x2a) to match any server's identity if the tlstmAddrServerFingerprint column is not blank. A row MUST NOT contain both a blank tlstmAddrServerFingerprint column and a '\*' in the tlstmAddrServerIdentity column since this would insecurely accept any presented certificate.

If there is no row in this table corresponding to an entry in the SNMP-TARGET-MIB and another certificate validation path algorithm (such as the path validation procedures defined in [[RFC5280](#)]) can be used, then the connection SHOULD still proceed."



```
::= { tlstmCertificateMapping 9 }
```

tlstmAddrEntry OBJECT-TYPE

SYNTAX TlstmAddrEntry

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 tlstmAddrEntry 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 a snmpTargetAddrEntry row then the connection SHOULD still proceed if some other certificate validation path algorithm (e.g. [RFC5280](#)) can be used."

INDEX { IMPLIED snmpTargetAddrName }

```
::= { tlstmAddrTable 1 }
```

TlstmAddrEntry ::= SEQUENCE {

tlstmAddrServerFingerprint Fingerprint,

tlstmAddrServerIdentity SnmpAdminString,

tlstmAddrStorageType StorageType,

tlstmAddrRowStatus RowStatus

}

tlstmAddrServerFingerprint OBJECT-TYPE

SYNTAX Fingerprint

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 { "" }

```
::= { tlstmAddrEntry 1 }
```

tlstmAddrServerIdentity OBJECT-TYPE

SYNTAX SnmpAdminString

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The reference identity to check against the identity



presented by the remote system. A single ASCII '\*' character (ASCII code 0x2a) may be used as a wildcard string and will match any presented server identity."

REFERENCE ["draft-saintandre-tls-server-id-check"](#)

DEFVAL { "\*" }

::= { tlstmAddrEntry 2 }

tlstmAddrStorageType 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 }

::= { tlstmAddrEntry 3 }

tlstmAddrRowStatus 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, a manager 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 tlstmAddrRowStatus column is 'notReady'.

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

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

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

An attempt to set these objects while the value of



```
    tlstmAddrRowStatus is active(1) will result in
    an inconsistentValue error."
 ::= { tlstmAddrEntry 4 }
```

```
-- *****
-- tlstmNotifications - Notifications Information
-- *****
```

#### tlstmServerCertificateUnknown NOTIFICATION-TYPE

OBJECTS { snmpTlstmSessionUnknownServerCertificate }

STATUS current

##### DESCRIPTION

"Notification that the server certificate presented by a 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 tlstmAddrTable or because no path could be found to known certificate authority.

To avoid notification loops, this notification MUST NOT be sent to servers that themselves have triggered the notification."

```
 ::= { tlstmNotifications 1 }
```

#### tlstmServerInvalidCertificate NOTIFICATION-TYPE

OBJECTS { tlstmAddrServerFingerprint,  
 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. I.E., a cryptographic validation occurred during certificate validation processing.

To avoid notification loops, this notification MUST NOT be sent to servers that themselves have triggered the notification."

```
 ::= { tlstmNotifications 2 }
```

```
-- *****
-- tlstmCompliances - Conformance Information
-- *****
```

```
tlstmCompliances OBJECT IDENTIFIER ::= { tlstmConformance 1 }
```

```
tlstmGroups OBJECT IDENTIFIER ::= { tlstmConformance 2 }
```





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

tlstmCompliance MODULE-COMPLIANCE
    STATUS          current
    DESCRIPTION
        "The compliance statement for SNMP engines that support the
        TLSTM-MIB"
    MODULE
        MANDATORY-GROUPS { tlstmStatsGroup,
                            tlstmIncomingGroup,
                            tlstmOutgoingGroup,
                            tlstmNotificationGroup }
    ::= { tlstmCompliances 1 }

-- *****
-- Units of conformance
-- *****

tlstmStatsGroup 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 which
        implements the SNMP TLS Transport Model."
    ::= { tlstmGroups 1 }

tlstmIncomingGroup OBJECT-GROUP
    OBJECTS {
        tlstmCertToTSNCount,
        tlstmCertToTSNTableLastChanged,
        tlstmCertToTSNFingerprint,
        tlstmCertToTSNMapType,
        tlstmCertToTSNData,
        tlstmCertToTSNStorageType,
        tlstmCertToTSNRowStatus
```



```
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects for maintaining
        incoming connection certificate mappings to
        tmSecurityNames of an SNMP engine which implements the
        SNMP TLS Transport Model."
    ::= { tlstmGroups 2 }

tlstmOutgoingGroup OBJECT-GROUP
    OBJECTS {
        tlstmParamsCount,
        tlstmParamsTableLastChanged,
        tlstmParamsClientFingerprint,
        tlstmParamsStorageType,
        tlstmParamsRowStatus,
        tlstmAddrCount,
        tlstmAddrTableLastChanged,
        tlstmAddrServerFingerprint,
        tlstmAddrServerIdentity,
        tlstmAddrStorageType,
        tlstmAddrRowStatus
    }
    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."
    ::= { tlstmGroups 3 }

tlstmNotificationGroup NOTIFICATION-GROUP
    NOTIFICATIONS {
        tlstmServerCertificateUnknown,
        tlstmServerInvalidCertificate
    }
    STATUS current
    DESCRIPTION
        "Notifications"
    ::= { tlstmGroups 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 the (D)TLS client and the (D)TLS server. 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 disconnect messages. Implementations SHOULD also have a system in place for detecting "broken" sessions through the use of heartbeats [[I-D.seggelmann-tls-dtls-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**

Most command responders have contextEngineIDs that are identical to the USM securityEngineID. USM provides a discovery service that allows command generators to determine a securityEngineID and thus a default contextEngineID to use. Because the TLS Transport Model does not make use of a securityEngineID, it may be difficult for command generators to discover a suitable default contextEngineID. Implementations should consider offering another engineID discovery mechanism to continue providing Command Generators with a suitable contextEngineID mechanism. A recommended discovery solution is documented 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 are additionally based on other transports (TCP, UDP and SCTP). These three 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](#)]. DTLS adds to the security considerations of TLS only because it is more vulnerable to denial of service attacks. A random cookie exchange was added to the handshake to prevent anonymous denial of service attacks. [RFC 4347](#) recommends that the cookie exchange is utilized for all handshakes. It is also RECOMMENDED by this specification that users enable this cookie exchange.

### **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 an configured accepted Certificate 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 responder'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 a 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 `tlstmCertToTSNTable` 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 `tlstmCertToTSNID` value.

## **9.2. 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.



### **9.3. 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:

- o The `tlstmParamsTable` can be used to change the outgoing X.509 certificate used to establish a (D)TLS connection. Modification to objects in this table need to be adequately authenticated since modification to 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.
- o The `tlstmAddrTable` can be used to change the expectations of the certificates presented by a remote (D)TLS server. Modification to objects in this table need to be adequately authenticated since modification to 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.
- o The `tlstmCertToTSNTable` is used to specify the mapping of incoming X.509 certificates to `tmSecurityNames` which eventually get mapped to a SNMPv3 `securityName`. Modification to objects in this table need to be adequately authenticated since modification to 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.

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

- o 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 also 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 is requested to assign:

1. a TCP port number above 1023 in the <http://www.iana.org/assignments/port-numbers> registry which will be the default port for receipt of SNMP command messages over a TLS Transport Model as defined in this document,
2. a TCP port number above 1023 in the <http://www.iana.org/assignments/port-numbers> registry which will be the default port for receipt of SNMP notification messages over a TLS Transport Model as defined in this document,
3. a UDP port number above 1023 in the <http://www.iana.org/assignments/port-numbers> registry which will be the default port for receipt of SNMP command messages over a DTLS/UDP connection as defined in this document,
4. a UDP port number above 1023 in the <http://www.iana.org/assignments/port-numbers> registry which will be the default port for receipt of SNMP notification messages over a DTLS/UDP connection as defined in this document,



5. a SCTP port number above 1023 in the <http://www.iana.org/assignments/port-numbers> registry which will be the default port for receipt of SNMP command messages over a DTLS/SCTP connection as defined in this document,
6. a SCTP port number above 1023 in the <http://www.iana.org/assignments/port-numbers> registry which will be the default port for receipt of SNMP notification messages over a DTLS/SCTP connection as defined in this document,
7. an SMI number under snmpDomains for the snmpTLSTCPDomain object identifier,
8. an SMI number under snmpDomains for the snmpDTLSUDPDDomain object identifier,
9. an SMI number under snmpDomains for the snmpDTLSSCTPDDomain object identifier,
10. a SMI number under snmpModules, for the MIB module in this document,
11. "tls" as the corresponding prefix for the snmpTLSTCPDomain in the SNMP Transport Model registry,
12. "dudp" as the corresponding prefix for the snmpDTLSUDPDDomain in the SNMP Transport Model registry,
13. "dsct" as the corresponding prefix for the snmpDTLSSCTPDDomain in the SNMP Transport Model registry;

If possible, IANA is requested to use matching port numbers for all assignments for SNMP Commands being sent over TLS, DTLS/UDP, DTLS/SCTP.

If possible, IANA is requested to use matching port numbers for all assignments for SNMP Notifications being sent over TLS, DTLS/UDP, DTLS/SCTP.

Editor's note: this section should be replaced with appropriate descriptive assignment text after IANA assignments are made and prior to publication.

## **11. Acknowledgements**

This document closely follows and copies the Secure Shell Transport Model for SNMP defined by David Harrington and Joseph Salowey in





[[RFC5292](#)].

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, Tom Petch, Randy Presuhn, Ray Purvis, Joseph Salowey, Jurgen Schonwalder, Dave Shield, 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 Hoogetboom, Dave Kornbau, Chris Knaian, Dan Knaul, Charles Limoges, Steve Moccaldi, Gerardo Orlando, and Brandon Yip.

## **[12.](#) References**

### **[12.1.](#) 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.
- [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.
- [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.

## **12.2. Informative References**

- [RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", [RFC 2522](#), March 1999.
- [RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", [RFC 3410](#), December 2002.
- [RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", [RFC 4306](#), December 2005.
- [RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", [RFC 4366](#), April 2006.



- [RFC5292] Chen, E. and S. Sangli, "Address-Prefix-Based Outbound Route Filter for BGP-4", [RFC 5292](#), August 2008.
- [RFC5343] Schoenwaelder, J., "Simple Network Management Protocol (SNMP) Context EngineID Discovery", [RFC 5343](#), September 2008.
- [I-D.saintandre-tls-server-id-check]  
Saint-Andre, P., Zeilenga, K., Hodges, J., and B. Morgan,  
"Best Practices for Checking of Server Identities in the  
Context of Transport Layer Security (TLS)".
- [I-D.seggelmann-tls-dtls-heartbeat]  
Seggellmann, R., Tuexen, M., and M. Williams, "Transport  
Layer Security and Datagram Transport Layer Security  
Heartbeat Extension".
- [AES] National Institute of Standards, "Specification for the  
Advanced Encryption Standard (AES)".
- [DES] National Institute of Standards, "American National  
Standard for Information Systems-Data Link Encryption".
- [DSS] National Institute of Standards, "Digital Signature  
Standard".
- [RSA] Rivest, R., Shamir, A., and L. Adleman, "A Method for  
Obtaining Digital Signatures and Public-Key  
Cryptosystems".
- [X509] , ITU., "INFORMATION TECHNOLOGY OPEN SYSTEMS  
INTERCONNECTION THE DIRECTORY: PUBLIC-KEY AND ATTRIBUTE  
CERTIFICATE FRAMEWORKS".

## [Appendix A.](#) (D)TLS Overview

The (D)TLS protocol is composed of two layers: the (D)TLS Record Protocol and the (D)TLS Handshake Protocol. The following subsections provide an overview of these two layers. Please refer to [\[RFC4347\]](#) for a complete description of the protocol.

### [A.1.](#) The (D)TLS Record Protocol

At the lowest layer, layered on top of the transport control protocol or a datagram transport protocol (e.g. UDP or SCTP) is the (D)TLS Record Protocol.



The (D)TLS Record Protocol provides security that has three basic properties:

- o The session can be confidential. Symmetric cryptography is used for data encryption (e.g., [[AES](#)], [[DES](#)] etc.). The keys for this symmetric encryption are generated uniquely for each session and are based on a secret negotiated by another protocol (such as the (D)TLS Handshake Protocol). The Record Protocol can also be used without encryption.
- o Messages can have data integrity. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA, MD5, etc.) are used for MAC computations. The Record Protocol can operate without a MAC, but is generally only used in this mode while another protocol is using the Record Protocol as a transport for negotiating security parameters.
- o Messages are protected against replay. (D)TLS uses explicit sequence numbers and integrity checks. DTLS uses a sliding window to protect against replay of messages within a session.

(D)TLS also provides protection against replay of entire sessions. In a properly-implemented keying material exchange, both sides will generate new random numbers for each exchange. This results in different encryption and integrity keys for every session.

## **[A.2.](#) The (D)TLS Handshake Protocol**

The (D)TLS Record Protocol is used for encapsulation of various higher-level protocols. One such encapsulated protocol, the (D)TLS Handshake Protocol, allows the server and client to authenticate each other and to negotiate an integrity algorithm, an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first octet of data. Only the (D)TLS client can initiate the handshake protocol. The (D)TLS Handshake Protocol provides security that has four basic properties:

- o The peer's identity can be authenticated using asymmetric (public key) cryptography (e.g., RSA [[RSA](#)], DSS [[DSS](#)], etc.). This authentication can be made optional, but is generally required by at least one of the peers.

(D)TLS supports three authentication modes: authentication of both the server and the client, server authentication with an unauthenticated client, and total anonymity. For authentication of both entities, each entity provides a valid certificate chain leading to an acceptable certificate authority. Each entity is responsible for verifying that the other's certificate is valid





and has not expired or been revoked. See [\[I-D.saintandre-tls-server-id-check\]](#) for further details on standardized processing when checking server certificate identities.

- o The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated handshake the secret cannot be obtained, even by an attacker who can place himself in the middle of the session.
- o The negotiation is not vulnerable to malicious modification: it is infeasible for an attacker to modify negotiation communication without being detected by the parties to the communication.
- o DTLS uses a stateless cookie exchange to protect against anonymous denial of service attacks and has retransmission timers, sequence numbers, and counters to handle message loss, reordering, and fragmentation.

## [Appendix B](#). PKIX Certificate Infrastructure

Users of a public key from a PKIX / X.509 certificate can be confident that the associated private key is owned by the correct remote subject (person or system) with which an encryption or digital signature mechanism will be used. This confidence is obtained through the use of public key certificates, which are data structures that bind public key values to subjects. The binding is asserted by having a trusted CA digitally sign each certificate. The CA may base this assertion upon technical means (i.e., proof of possession through a challenge-response protocol), presentation of the private key, or on an assertion by the subject. A certificate has a limited valid lifetime which is indicated in its signed contents. Because a certificate's signature and timeliness can be independently checked by a certificate-using client, certificates can be distributed via untrusted communications and server systems, and can be cached in unsecured storage in certificate-using systems.

ITU-T X.509 (formerly CCITT X.509) or ISO/IEC/ITU 9594-8 [\[X509\]](#), which was first published in 1988 as part of the X.500 Directory recommendations, defines a standard certificate format which is a certificate which binds a subject (principal) to a public key value. This was later further expanded and documented in [\[RFC5280\]](#).

A X.509 certificate is a sequence of three required fields:



**tbsCertificate:** The tbsCertificate field contains the names of the subject and issuer, a public key associated with the subject, a validity period, and other associated information. This field may also contain extension components.

**signatureAlgorithm:** The signatureAlgorithm field contains the identifier for the cryptographic algorithm used by the certificate authority (CA) to sign this certificate.

**signatureValue:** The signatureValue field contains a digital signature computed by the CA upon the ASN.1 DER encoded tbsCertificate field. The ASN.1 DER encoded tbsCertificate is used as the input to the signature function. This signature value is then ASN.1 DER encoded as a BIT STRING and included in the Certificate's signature field. By generating this signature, the CA certifies the validity of the information in the tbsCertificate field. In particular, the CA certifies the binding between the public key material and the subject of the certificate.

The basic X.509 authentication procedure is as follows: A system is initialized with a number of root certificates that contain the public keys of a number of trusted CAs. When a system receives a X.509 certificate, signed by one of those CAs, the certificate has to be verified. It first checks the signatureValue field by using the public key of the corresponding trusted CA. Then it compares the digest of the received certificate with a digest of the tbsCertificate field. If they match, then the subject in the tbsCertificate field is authenticated.

## [Appendix C](#). Target and Notification Configuration Example

Configuring the SNMP-TARGET-MIB and NOTIFICATION-MIB along with access control settings for the SNMP-VIEW-BASED-ACM-MIB can be a daunting task without an example to follow. The following section describes an example of what pieces must be in place to accomplish this configuration.

The isAccessAllowed() ASI requires configuration to exist in the following SNMP-VIEW-BASED-ACM-MIB tables:

```
vacmSecurityToGroupTable
vacmAccessTable
vacmViewTreeFamilyTable
```

The only table that needs to be discussed as particularly different here is the vacmSecurityToGroupTable. This table is indexed by both the SNMPv3 security model and the security name. The security model,



when TLSTM is in use, should be set to the value of 4, corresponding to the TSM [[RFC5591](#)]. An example vacmSecurityToGroupTable row might be filled out as follows (using a single SNMP SET request):

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

This example will assume that the "administrators" group has been given proper permissions via rows in the vacmAccessTable and vacmViewTreeFamilyTable.

Depending on whether this VACM configuration is for a Command Responder or a command generator the security name "blueberry" will come from a few different locations.

### **C.1. Configuring the Notification Originator**

For notification originators performing authorization checks, the server's certificate must be verified against the expected certificate before proceeding to send the notification. The expected certificate from the server may be listed in the tlstmAddrTable or may be determined through other X.509 path validation mechanisms. The securityName to use for VACM authorization checks is set by the SNMP-TARGET-MIB's snmpTargetParamsSecurityName column.

The certificate that the notification originator should present to the server is taken from the tlstmParamsClientFingerprint column from the appropriate entry in the tlstmParamsTable table.

### **C.2. Configuring the Command Responder**

For command responder applications, the vacmSecurityName "blueberry" value is a value that derived from an incoming (D)TLS session. The mapping from a received (D)TLS client certificate to a tmSecurityName is done with the tlstmCertToTSNTable. The certificates must be loaded into the device so that a tlstmCertToTSNEntry may refer to it. As an example, consider the following entry which will provide a mapping from a client's public X.509's hash fingerprint directly to the "blueberry" tmSecurityName:



```
tlstmCertToTSNID          = 1      (chosen by ordering preference)
tlstmCertToTSNFingerprint = HASH   (appropriate fingerprint)
tlstmCertToTSNMapType      = 1      (specified)
tlstmCertToTSNSecurityName = "blueberry"
tlstmCertToTSNStorageType  = 3      (nonVolatile)
tlstmCertToTSNRowStatus    = 4      (createAndGo)
```

The above is an example of how to map a particular certificate to a particular tmSecurityName. It is recommended, however, that users make use of direct subjectAltName or CommonName mappings where possible as it provides a more scalable approach to certificate management. This entry provides an example of using a subjectAltName mapping:

```
tlstmCertToTSNID          = 1      (chosen by ordering preference)
tlstmCertToTSNFingerprint = HASH   (appropriate fingerprint)
tlstmCertToTSNMapType      = 2      (bySubjectAltName)
tlstmCertToTSNSANType      = 1      (any)
tlstmCertToTSNStorageType  = 3      (nonVolatile)
tlstmCertToTSNRowStatus    = 4      (createAndGo)
```

The above entry indicates the subjectAltName field for certificates created by an issuing certificate with a corresponding fingerprint will be trusted to always produce common names that are directly one-to-one mappable into tmSecurityNames. This type of configuration should only be used when the certificate authorities naming conventions are carefully controlled.

In the example, if the incoming (D)TLS client provided certificate contained a subjectAltName where the first listed subjectAltName in the extension is the rfc822Name of "blueberry@example.com", the certificate was signed by a certificate matching the tlstmCertToTSNFingerprint value and the CA's certificate was properly installed on the device then the string "blueberry@example.com" would be used as the tmSecurityName for the session.

#### Author's Address

Wes Hardaker  
Sparta, Inc.  
P.O. Box 382  
Davis, CA 95617  
USA

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



