ISMS
Internet-Draft
Intended status: Standards Track
Expires: March 5, 2010

W. Hardaker
Sparta, Inc.

September 1, 2009

Transport Layer Security Transport Model for SNMP
draft-ietf-isms-dtls-tm-00.txt

Status of this Memo

Copyright Notice

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. 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.

Table of Contents

1. Introduction 5
1.1. Conventions 7
2. The Transport Layer Security Protocol 8
2.1. SNMP requirements of (D)TLS 8
3. How the TLSTM fits into the Transport Subsystem 8
3.1. Security Capabilities of this Model 10
3.1.1. Threats 10
3.1.2. Message Protection 12
3.1.3. (D)TLS Sessions 13
3.2. Security Parameter Passing 13
3.3. Notifications and Proxy 14
4. Elements of the Model 15
4.1. X.509 Certificates 15
4.1.1. Provisioning for the Certificate 15
4.2. Messages 16
4.3. SNMP Services 16
4.3.1. SNMP Services for an Outgoing Message 16
4.3.2. SNMP Services for an Incoming Message 17
4.4. (D)TLS Services 18
4.4.1. Services for Establishing a Session 18
4.4.2. (D)TLS Services for an Incoming Message 20
4.4.3. (D)TLS Services for an Outgoing Message 20
4.5. Cached Information and References 21
4.5.1. TLS Transport Model Cached Information 21
5. Elements of Procedure 21
5.1. Procedures for an Incoming Message 22
5.1.1. DTLS Processing for Incoming Messages 22
5.1.2. Transport Processing for Incoming Messages 23
5.2. Procedures for an Outgoing Message 25
5.3. Establishing a Session 26
5.4. Closing a Session 28
6. MIB Module Overview 29
6.1. Structure of the MIB Module 29
6.2. Textual Conventions 29
6.3. Statistical Counters 29
6.4. Configuration Tables 29
6.5. Relationship to Other MIB Modules 30
6.5.1. MIB Modules Required for IMPORTS 30
7. MIB Module Definition 30
8. Operational Considerations 53
8.1. Sessions 53
8.2. Notification Receiver Credential Selection 53
8.3. contextEngineID Discovery 54
9. Security Considerations 54
9.1. Certificates, Authentication, and Authorization 54
9.2. Use with SNMPv1/SNMPv2c Messages 55

9.3. MIB Module Security 56
10. IANA Considerations 57
11. Acknowledgements 58
12. References 59
12.1. Normative References 59
12.2. Informative References 60
Appendix A. (D)TLS Overview 61
A.1. The (D)TLS Record Protocol 61
A.2. The (D)TLS Handshake Protocol 62
Appendix B. PKIX Certificate Infrastructure 63
Appendix C. Target and Notificaton Configuration Example 64
Author's Address 66

1. Introduction

It is important to understand the modular SNMPv3 architecture as defined by [RFC3411] and enhanced by the Transport Subsystem [I-D.ietf-isms-tmsm]. 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 [I-D.ietf-isms-tmsm]. DTLS is the datagram variant of the Transport Layer Security (TLS) protocol [RFC5246]. The Transport Model in this document is referred to as the Transport Layer Security Transport Model (TLSTM). TLS and DTLS take advantage of the X.509 public keying infrastructure [RFC5280]. This transport model is designed to meet the security and operational needs of network administrators, operate 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 keying infrastructures.

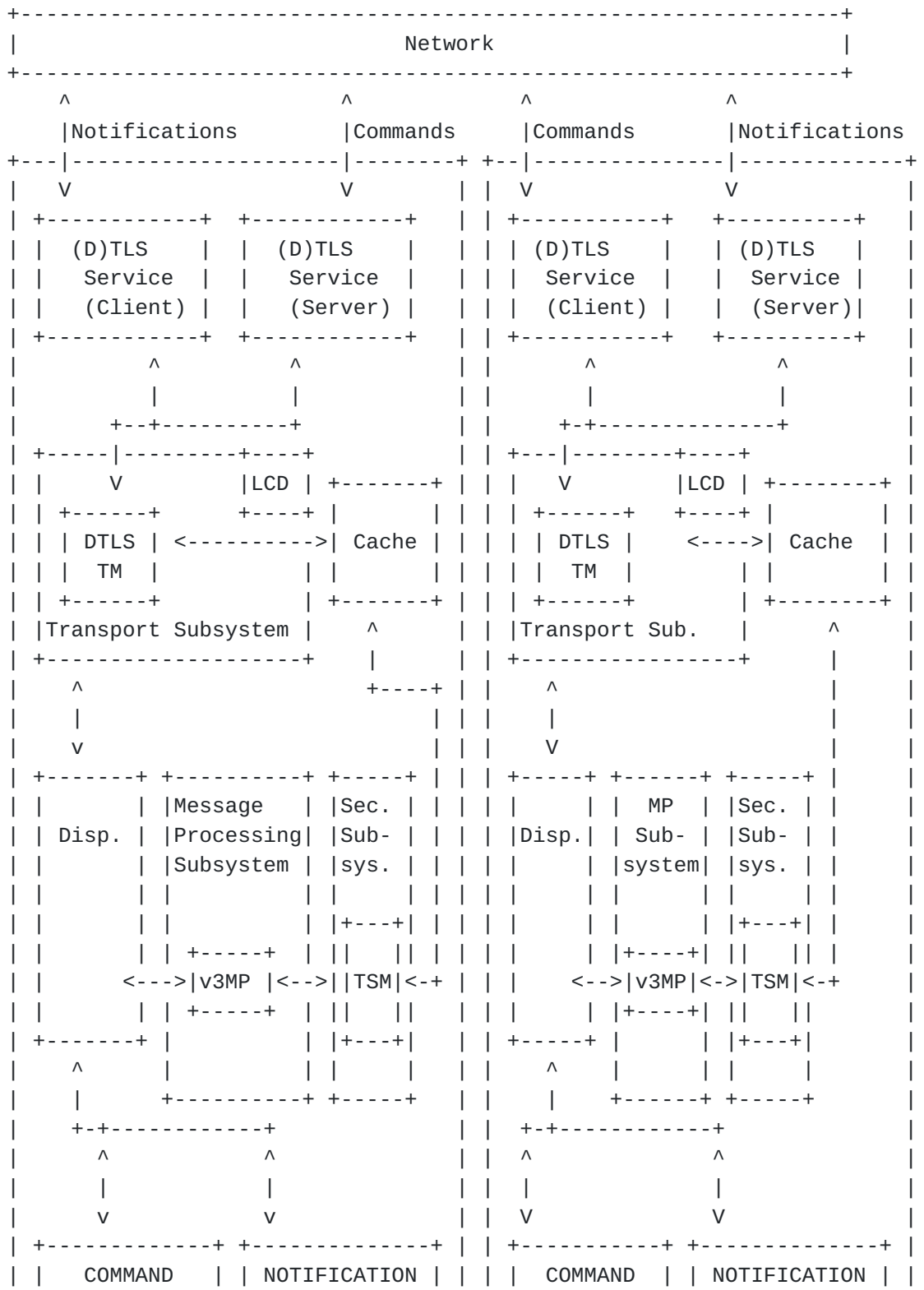
This document also defines a portion of the Management Information Base (MIB) for use with network management protocols. In particular it defines objects for managing the TLS Transport Model for SNMP.

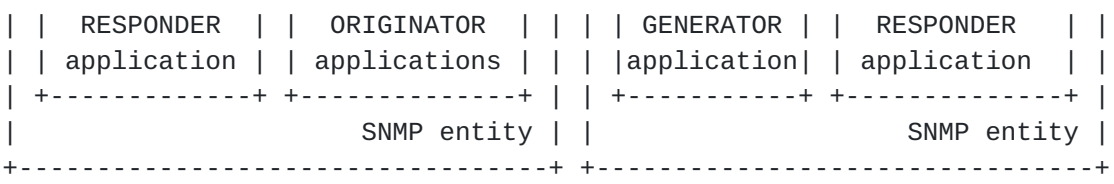
For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of RFC [RFC3410].

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





1.1. Conventions

For consistency with SNMP-related specifications, this document favors terminology as defined in STD62 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.

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.

The terms "manager" and "agent" are not used in this document, because in the RFC 3411 architecture [RFC3411], all SNMP entities have the capability of acting in either manager or agent or in both roles 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.

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. Either SNMP entity may act as client or as server, as discussed further below.

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 [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 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, source 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 [RFC4347] for a complete description of the protocol.

2.1. SNMP requirements of (D)TLS

To properly support the SNMP over TLS Transport Model, the (D)TLS implementation requires the following:

- The TLS Transport Model SHOULD always use authentication of both the server and the client.
- At a minimum the TLS Transport Model MUST support authentication of the Command Generator principals to guarantee the authenticity of the securityName.
- The TLS Transport Model SHOULD support the message encryption to protect sensitive data from eavesdropping attacks.

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

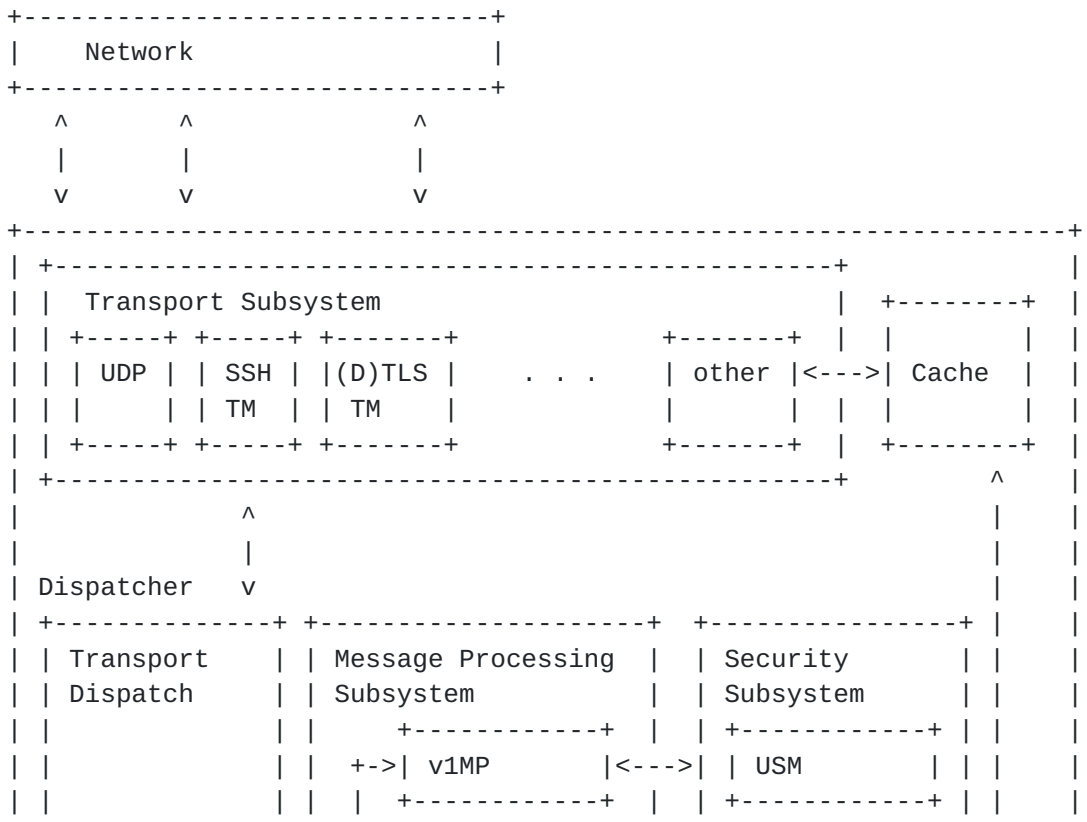
layer and the message dispatcher [RFC3411] component of the SNMP engine and the Transport Subsystem.

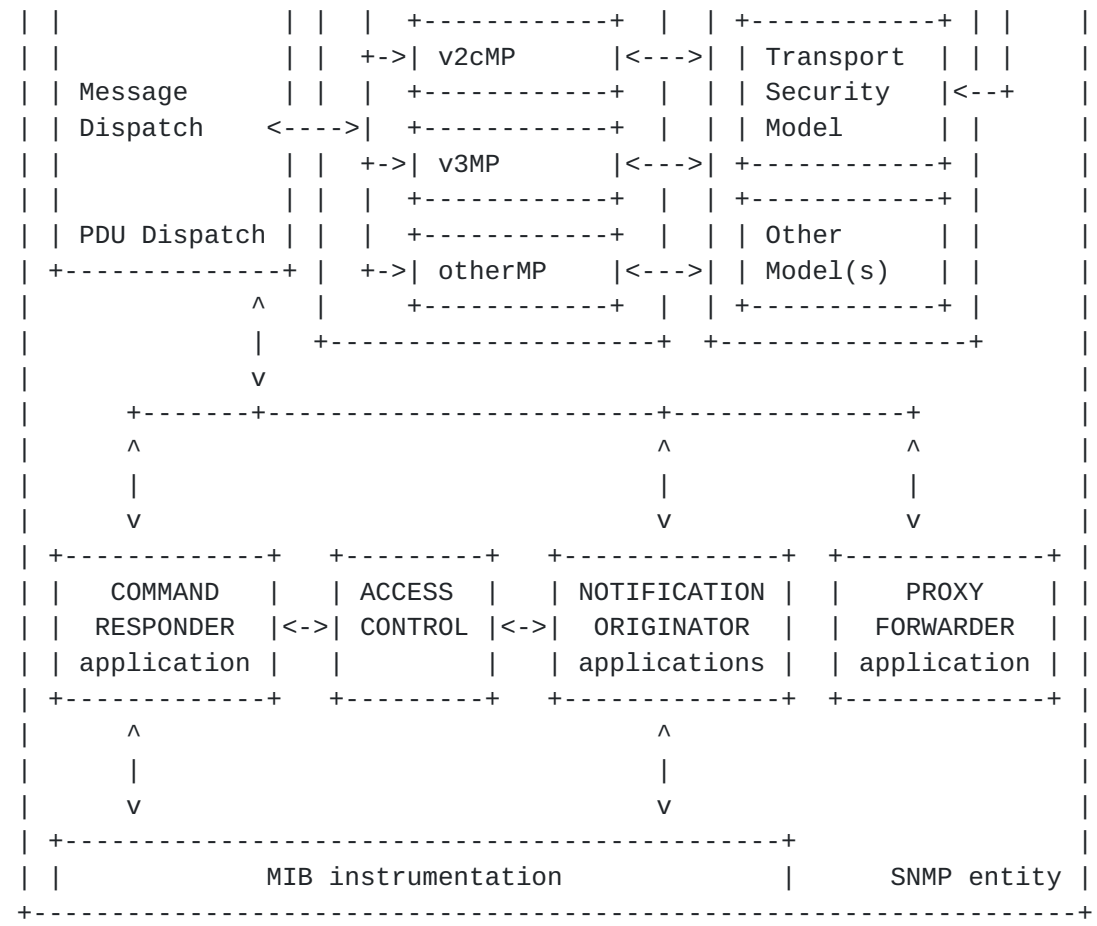
The TLS Transport Model will establish a session between itself and the TLS Transport Model of another SNMP engine. The sending transport model passes unprotected messages from the dispatcher to (D)TLS to be protected, and the receiving transport model accepts decrypted and authenticated/integrity-checked 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 SHOULD 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:





3.1. Security Capabilities of this Model

3.1.1. Threats

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

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

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

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

   The TLSTM provides for authentication of the Command Generator, Command Responder, Notification Generator, Notification Responder and Proxy Forwarder through the use of X.509 certificates.

   The masquerade threat can be mitigated against by using an appropriate Access Control Model (ACM) such as the View-based Access Control Module (VACM) [RFC3415]. In addition, it is important to authenticate and verify both the authenticated identity of the (D)TLS client and the (D)TLS server to protect against this threat. (See Section 9 for more detail.)

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 for replay protection, see [RFC4347]. The technique used is similar to that as in IPsec AH/ESP [RFC4302] [RFC4303], by maintaining a bitmap window of received records. Records that are too old to fit in the window and records that have previously been received are silently discarded. The replay detection feature is optional, since packet duplication can also occur naturally due to routing errors and does not necessarily indicate an active attack. Applications may conceivably detect duplicate packets and accordingly modify their data transmission strategy.

4. Disclosure - The disclosure threat is the danger of eavesdropping on the exchanges between SNMP engines. Protecting against this threat may be required by local policy at the deployment site.

   Symmetric cryptography (e.g., AES [AES], DES [DES] etc.) can be used by (D)TLS for data privacy. 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).

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 it is more vulnerable to spoofed messages.

   In order to counter both of 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 handshakes but in environments where amplification could be an issue or has been detected it is RECOMMENDED that the cookie exchange is utilized.

3.1.2. Message Protection

The RFC 3411 architecture recognizes three levels of security:

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

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

When an application requests a session for a message, through the cache, the application 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. However, different port numbers will need to be used by at least one side of the connection to differentiate between the (D)TLS sessions. This is the only way to ensured proper selection of a session ID for an incoming (D)TLS message.

The authentication, integrity and privacy algorithms used by the (D)TLS Protocol [RFC4347] 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 and 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, securityName, and requestedSecurityLevel associated with each session. Each unique combination of these parameters MUST have a locally-chosen unique dtlsSessionID associated for active sessions. For further information see Section 4.4 and Section 5. TLS and DTLS over SCTP sessions, on the other hand, do not require a unique paring of attributes since their lower layer protocols (TCP and SCTP) already provide adequate session framing.

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 (i.e., securityLevel, securityName, 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 command generators or notification originators. Command generators are frequently operated by a human, but notification originators are usually unmanned automated processes. The targets to whom notifications 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 Generator, 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 snmpTLSDomain, snmpDTLSUDPDomain, or snmpDTLSSCTPDomain object and an appropriate snmpTLSAddress, snmpDTLSUDPAddress or snmpDTLSSCTPAddress value respectively. The snmpTargetParamsMPModel column of the snmpTargetParamsTable should be set to a value of 3 to indicate the SNMPv3 message processing model. The snmpTargetParamsSecurityName should be set to an appropriate securityName value and the tlstmParamsClientHashType and tlstmParamsClientHashValue parameters of the tlstmParamsTable should be set to values that refer to a locally held certificate to be used. The tlstmAddrServerHashType and tlstmAddrServerHashValue must be set to a hash value that refers to a locally held copy of the server's presented identity certificate. Other parameters, for example cryptographic configuration such as cipher suites to use, must come from configuration mechanisms not defined in this document. The

other needed configuration may be configured using SNMP or other implementation-dependent mechanisms (for example, via a CLI). This 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 makes use of X.509 certificates for authentication of both sides of the transport. This section discusses the use of certificates in the TLSTM. An overview of X.509 certificate infrastructure can be found in Appendix B.

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. 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.

The authenticated tmSecurityName of the principal is looked up using the tlstmCertToSNTable. This table either:

- Maps a certificate's fingerprint hash type and value to a directly specified tmSecurityName.

- Identifies a certificate issuer's fingerprint hash type and value and allows child certificate's subjectAltName or CommonName to directly used as the tmSecurityNome.

The certificate trust anchors, being either CA certificates or public keys for use by self-signed certificates, must be installed through an out of band trusted mechanism into the server and its authenticity MUST be verified before access is granted. Implementations SHOULD choose to discard any connections for which no potential tlstmCertToSNTable mapping exists before performing certificate verification to avoid expending computational resources associated with certificate verification.

The typical enterprise configuration will map a "subjectAltName" component of the tbsCertificate to the TLSTM specific tmSecurityName. Thus, the authenticated identity can be obtained by the TLS Transport Model by extracting the subjectAltName(s) from the peer's certificate and the receiving application will have an appropriate tmSecurityName for use by components like an access control model. This setup requires very little configuration: a single row in the tlstmCertToSNTable referencing a certificate authority.

An example 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, may perform an identity mapping or a more complex mapping to derive the securityName from the tmSecurityName offered by the TLS Transport Model.

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 (D)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 passed as parameters in 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.

destTransportDomain: The transport domain for the associated destTransportAddress. The Transport Model uses this parameter to determine the transport type of the associated destTransportAddress. This parameter may also be used by the transport subsystem to route the message to the appropriate Transport Model. This document specifies three TLS and DTLS based Transport Domains for use: the snmpTLSDomain, the snmpDTLSUDPDomain and the snmpDTLSSCTPDomain.

destTransportAddress: The transport address of the destination TLS Transport Model in a format specified by the SnmpTLSAddress, the SnmpDTLSUDPAddress or the SnmpDTLSSCTPAddress TEXTUAL-CONVENTIONs.

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

outgoingMessageLength: The length of the outgoing message.

tmStateReference: A handle/reference to tmSecurityData 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 passed as parameters in 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 three TLS and DTLS based Transport Domains for use: the snmpTLSDomain, the snmpDTLSUDPDomain and the snmpDTLSSCTPDomain.

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

incomingMessage: The whole SNMP message stripped of all (D)TLS protection data.

incomingMessageLength: The length of the SNMP message after being processed by (D)TLS.

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

4.4. (D)TLS Services

This section describes the services provided by the (D)TLS Transport Model with their inputs and outputs. These services are between the TLS Transport Model and the (D)TLS transport layer. The following sections describe services for establishing and closing a session and for passing messages between the (D)TLS transport layer and the TLS Transport Model.

4.4.1. Services for Establishing a Session

The TLS Transport Model provides the following ASI to describe the data passed between the Transport Model and the (D)TLS transport layer for session establishment.

```
statusInformation =          -- errorIndication or success
openSession(
IN   destTransportDomain     -- transport domain to be used
IN   destTransportAddress    -- transport address to be used
IN   securityName            -- on behalf of this principal
IN   securityLevel           -- Level of Security requested
OUT  tlsSessionID            -- Session identifier for (D)TLS
)
```

The abstract data elements passed as parameters in the abstract

service primitives are as follows:

statusInformation: An indication of whether the process was successful or not. If not, then the status information will include the error indication provided by (D)TLS.

destTransportDomain: The transport domain for the associated destTransportAddress. The TLS Transport Model uses this parameter to determine the transport type of the associated destTransportAddress. This document specifies three TLS and DTLS based Transport Domains for use: the snmpTLSDomain, the snmpDTLSUDPDomain, and the snmpDTLSSCTPDomain.

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

securityName: The security name representing the principal on whose behalf the message will be sent.

securityLevel: The level of security requested by the application.

dtlsSessionID: An implementation-dependent session identifier to reference the specific (D)TLS session.

DTLS and UDP do not provide a session de-multiplexing mechanism and it is possible that implementations will only be able to identify a unique session based on a unique combination of source address, destination address, source UDP port number and destination UDP port number. Because of this, when establishing a new sessions implementations MUST use a different UDP source port number for each connection to a remote destination IP-address/port-number combination to ensure the remote entity can properly disambiguate between multiple sessions from a host to the same port on a server. TLS and DTLS over SCTP provide session de-multiplexing so this restriction is not needed for TLS or DTLS over SCTP implementations.

The procedural details for establishing a session are further described in Section 5.3.

Upon completion of the process the TLS Transport Model returns status information and, if the process was successful the dtlsSessionID. Other implementation-dependent data from (D)TLS are also returned. The dtlsSessionID is stored in an implementation- dependent manner and tied to the tmSecurityData for future use of this session.

4.4.2. (D)TLS Services for an Incoming Message

When the TLS Transport Model invokes the (D)TLS record layer to verify proper security for the incoming message, it must use the following ASI:

```
statusInformation =          -- errorIndication or success
tlsRead(
IN   tlsSessionID            -- Session identifier for (D)TLS
IN   wholeTlsMsg             -- as received on the wire
IN   wholeTlsMsgLength       -- length as received on the wire
OUT  incomingMessage         -- the whole SNMP message from (D)TLS
OUT  incomingMessageLength   -- the length of the SNMP message
)
```

The abstract data elements passed as parameters in the abstract service primitives are as follows:

statusInformation: An indication of whether the process was successful or not. If not, then the status information will include the error indication provided by (D)TLS.

tlsSessionID: An implementation-dependent session identifier to reference the specific (D)TLS session. How the (D)TLS session ID is obtained for each message is implementation-dependent. As an implementation hint, for dtls over udp the TLS Transport Model can examine incoming messages to determine the source IP address, source port number, destination IP address, and destination port number and use these values to look up the local tlsSessionID in the list of active sessions.

wholeDtlsMsg: The whole message as received on the wire.

wholeDtlsMsgLength: The length of the message as it was received on the wire.

incomingMessage: The whole SNMP message stripped of all (D)TLS privacy and integrity data.

incomingMessageLength: The length of the SNMP message stripped of all (D)TLS privacy and integrity data.

4.4.3. (D)TLS Services for an Outgoing Message

When the TLS Transport Model invokes the (D)TLS record layer to encapsulate and transmit a SNMP message, it must use the following ASI.

```
statusInformation =          -- errorIndication or success
tlsWrite(
IN   tlsSessionID            -- Session identifier for (D)TLS
IN   outgoingMessage         -- the message to send
IN   outgoingMessageLength   -- its length
)
```

The abstract data elements passed as parameters in the abstract service primitives are as follows:

statusInformation: An indication of whether the process was successful or not. If not, then the status information will include the error indication provided by (D)TLS.

tlsSessionID: An implementation-dependent session identifier to reference the specific (D)TLS session that the message should be sent using.

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

outgoingMessageLength: The length of the outgoing message.

4.5. 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" [I-D.ietf-isms-tmsm] defines general requirements for caches and references.

4.5.1. TLS Transport Model Cached Information

The TLSTM has no specific responsibilities regarding the cached information beyond those discussed in "Transport Subsystem for the Simple Network Management Protocol" [I-D.ietf-isms-tmsm]

5. Elements of Procedure

Abstract service interfaces have been defined by RFC 3411 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. These RFC 3411-defined data flows are referred to here as public interfaces.

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 first prior to being released.

An error indication may return an OID and value for an incremented counter if the information is available at the point where the error is detected.

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 steps are broken into two different sections. Section 5.1.1 describes the needed steps for de-multiplexing multiple DTLS sessions, which is specifically needed for DTLS over UDP sessions. Section 5.1.2 describes the steps specific to transport processing once the (D)TLS processing has been completed.

5.1.1. DTLS Processing for Incoming Messages

DTLS is significantly different in terms of session handling than TCP-based session streams like SSH or TLS. The DTLS protocol, which is datagram-based, does not have a session identifier when run over UDP that allows implementations to determine through what session a packet arrived. DTLS over SCTP and TLS over TCP streams have built in session demultiplexing and thus the steps in this section are not necessary for those protocol combinations. It is always critical, however, that implementations be able to derive a tlsSessionID from any demultiplexing process.

A process for de-multiplexing multiple DTLS sessions arriving over UDP must be incorporated into the procedures for processing an incoming message. The steps in this section describe how this can be accomplished, although any implementation dependent method should be suitable as long as the results are consistently 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 (tlsSessionID).

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 a implementation-dependent tlsSessionID.

1) The TLS Transport Model examines the raw UDP message, in an implementation-dependent manner. If the message is not a DTLS message then it should be discarded. If the message is not a (D)TLS Application Data message then the message should be processed by the underlying DTLS framework as it is (for example) a session initialization or session modification message and no further steps below should be taken by the DTLS Transport.

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 and its tlsSessionID.

3) If a matching entry in the LCD does not exist then the message is discarded. Increment the tlstmSessionNoAvailableSessions 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 (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 "rogue" message was routed to the SNMP entity by mistake. An entry might also be missing because of a "broken" session (see operational considerations).

4) Retrieve the tlsSessionID from the LCD.

5) The tlsWholeMsg, and the tlsSessionID are passed to DTLS for integrity checking and decryption using the tlsRead() ASI.

6) If the message fails integrity checks or other (D)TLS security processing then the tlstmDTLSProtectionErrors counter is incremented, the message is discarded and processing of the message is stopped.

7) The output of the tlsRead results in an incomingMessage and an incomingMessageLength. These results and the tlsSessionID are used below in the Section 5.1.2 to complete the processing of the incoming message.

5.1.2. Transport Processing for Incoming 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.

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

tmTransportDomain = snmpTLSDomain, snmpDTLSUDPDomain or snmpDTLSSCTPDomain as appropriate.

tmTransportAddress = The address the message originated from, determined in an implementation dependent way.

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 and Section 5.3. This value MUST be constant during the lifetime of the (D)TLS session.

tmSessionID = The tlsSessionID, which MUST be a unique session identifier for this (D)TLS session. 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 tlsSessionID discussed in Section 5.1.1. tmSessionID refers to the session identifier when stored in the tmStateReference and tlsSessionID refers to the session identifier when stored in the LCD. They MUST always be equal when processing a given session's traffic.

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

The TLS Transport Model passes the transportDomain, transportAddress, wholeMessage, and wholeMessageLength to the dispatcher using the receiveMessage ASI:

```
    statusInformation =
    receiveMessage(
    IN   transportDomain     -- snmpTLSDomain, snmpDTLSUDPDomain,
                             -- or snmpDTLSSCTPDomain
    IN   transportAddress    -- address for the received message
    IN   wholeMessage        -- the whole SNMP message from (D)TLS
    IN   wholeMessageLength  -- the length of the SNMP message
    IN   tmStateReference    -- (NEW) transport info
     )
```

5.2. Procedures for an Outgoing 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             -- (NEW) 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) Extract tmSessionID, tmTransportAddress, tmSecurityName, tmRequestedSecurityLevel. and tmSameSecurity from the tmStateReference. Note: The tmSessionID value may be undefined if session exists yet.

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

3) 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 as described below in step 4b. An implementation MAY choose to return an error to the calling module.

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

   4a) If there is a corresponding LCD entry, then this session will be used to send the message.

   4b) If there is not a corresponding LCD entry, then open a session using the openSession() ASI (discussed further in Section 4.4.1). 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, increment the tlstmSessionOpenErrors, and return an error indication to

the calling module.

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

5.3. Establishing a Session

The TLS Transport Model provides the following primitive to establish a new (D)TLS session (previously discussed in Section 4.4.1):

```
statusInformation =          -- errorIndication or success
openSession(
IN   destTransportDomain     -- transport domain to be used
IN   destTransportAddress    -- transport address to be used
IN   securityName            -- on behalf of this principal
IN   securityLevel           -- Level of Security requested
OUT  tlsSessionID            -- Session identifier for (D)TLS
)
```

The following describes the procedure to follow to establish a SNMP over (D)TLS session between SNMP engines to exchange SNMP messages. This process is followed by any SNMP engine establishing a session for subsequent use.

This MAY be done automatically for SNMP messages which are not Response or Report messages.

(D)TLS provides no explicit manner for transmitting an identity the client wishes to connect to during or prior to key exchange to facilitate certificate selection at the server (e.g. at a Notification Receiver). I.E., there is no available mechanism for sending notifications to a specific principal at a given TCP, UDP or SCTP port. Therefore, implementations MAY support responding with multiple identities using separate TCP, UDP or SCTP port numbers to indicate the desired principal or some other implementation-dependent solution.

1) 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 should be 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.

2) 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 tlstmSessionOpenErrors is incremented, an error indication is returned, and session establishment processing stops.

3) Once the secure session is established and both sides have been authenticated, certificate validation and identity expectations are performed.

   a) The (D)TLS server side of the connection identifies the authenticated identity from the (D)TLS client's principal certificate using appropriate certificate path validation procedures (e.g. [RFC5280]) using configuration information from the tlstmCertToSNTable mapping table. The resulting derived securityName is recorded in the tmStateReference cache as tmSecurityName. The details of the lookup process are fully described in the DESCRIPTION clause of the tlstmCertToSNTable MIB object. If this verification fails in any way (for example because of failures in cryptographic verification or the lack of an appropriate row in the tlstmCertToSNTable) then the session establishment MUST fail, the tlstmSessionInvalidClientCertificates object is incremented and processing is stopped.

   b) The (D)TLS client side of the connection SHOULD verify that authenticated identity of the (D)TLS server's certificate is the certificate expected. This can be done using the configuration found in the tlstmAddrTable if the client is establishing the connection based on SNMP-TARGET-MIB configuration. The client MUST always perform appropriate certificate path validation procedures (e.g. [RFC5280]) to ensure the certificate is cryptographically valid.

If strong authentication is desired then the (D)TLS server address or naming mechanism MUST always be verified against the certificate's contents. Methods for doing this are described in [I-D.saintandre-tls-server-id-check]. Matching the server's naming against SubjectAltName extension values is the preferred mechanism for comparison, but matching the CommonName MAY be used.

(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 was not the expected identity) then the session establishment MUST fail, the tlstmSessionInvalidServerCertificates object is incremented and processing is stopped.

4) The (D)TLS-specific session identifier is passed to the TLS Transport Model and associated with the tmStateReference cache entry to indicate that the session has been established successfully and to point to a specific (D)TLS session for future use.

5.4. Closing a Session

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

```
statusInformation =
closeSession(
IN  tmStateReference        -- transport info
)
```

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) Look up the session in the cache and the LCD using the tmStateReference.

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

3) Delete the entry from the cache and any other implementation-dependent information in the LCD.

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 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:

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

- Public certificate fingerprint Hashing Types and Values. These textual conventions allow for MIB module objects to refer generically to a stored X.509 certificate using a simple hash value as a reference pointer.

6.3. Statistical Counters

The TLSTM-MIB defines some statical counters that can provide network managers with feedback 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 help in configuring a MIB-instrumented device for sending and receiving SNMP messages over (D)TLS. In particular, there is a MIB table that extends the SNMP-TARGET-MIB for configuring certificates to be used and a MIB table for mapping incoming (D)TLS client certificates to securityNames.

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 SNMP-CONF [RFC2580].

7. MIB Module Definition

```
TLSTM-MIB DEFINITIONS ::= BEGIN

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

tlstmMIB MODULE-IDENTITY
    LAST-UPDATED "200807070000Z"
    ORGANIZATION " "
    CONTACT-INFO "WG-EMail:
                  Subscribe:

                  Chairs:
                  Co-editors:
                                "
```

```
    DESCRIPTION  "The TLS Transport Model MIB

                 Copyright (C) The IETF Trust (2008). 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     "200807070000Z"
       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 SNMP-DTLS-TM-MIB
-- ************************************************

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

-- ************************************************
-- Objects
-- ************************************************

snmpTLSDomain 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
        snmpTLSDomain is 'tls'.  This prefix may be used by
        security models or other components to identify what 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.
```

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

        When an SNMP entity uses the snmpDTLSUDPDomain transport
        model, it must be capable of accepting messages up to
        the maximum MTU size for an interface it supports, minus the
        needed IP, UDP, DTLS and other protocol overheads.

        The securityName prefix to be associated with the
        snmpDTLSUDPDomain is 'dudp'.  This prefix may be used by
        security models or other components to identify what 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 'dtls' is not assigned to this document.

snmpDTLSSCTPDomain OBJECT-IDENTITY
    STATUS      current
    DESCRIPTION
        "The SNMP over DTLS/SCTP transport domain. The corresponding
        transport address is of type SnmpDTLSSCTPAddress.

        When an SNMP entity uses the snmpDTLSSCTPDomain transport
        model, it must be capable of accepting messages up to
        the maximum MTU size for an interface it supports, minus the
        needed IP, SCTP, DTLS and other protocol overheads.

        The securityName prefix to be associated with the
        snmpDTLSSCTPDomain is 'dsct'.  This prefix may be used by
        security models or other components to identify what 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 'dtls' is not assigned to this document.

SnmpTLSAddress ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1a"
    STATUS       current
    DESCRIPTION
        "Represents a TCP connection address for an IPv4 address, an
        IPv6 address or an ASCII encoded host name and port number.

        The hostname must be encoded in ASCII, as specified in RFC3490
        (Internationalizing Domain Names in Applications) followed by
        a colon ':' (ASCII character 0x3A) and a decimal port number
        in ASCII. The name SHOULD be fully qualified whenever
        possible.

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

        An IPv6 address must be a colon separated format, surrounded
        by square brackets (ASCII characters 0x5B and 0x5D), followed
        by a colon ':' (ASCII character 0x3A) and a decimal port
        number in ASCII.

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

SnmpDTLSUDPAddress ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1a"
    STATUS       current
    DESCRIPTION
        "Represents a UDP connection address for an IPv4 address, an
        IPv6 address or an ASCII encoded host name and port number.

        The hostname must be encoded in ASCII, as specified in RFC3490
        (Internationalizing Domain Names in Applications) followed by
        a colon ':' (ASCII character 0x3A) and a decimal port number
        in ASCII. The name SHOULD be fully qualified whenever
        possible.

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

        An IPv6 address must be a colon separated format, surrounded
        by square brackets (ASCII characters 0x5B and 0x5D), followed
        by a colon ':' (ASCII character 0x3A) and a decimal port
        number in ASCII.

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

SnmpDTLSSCTPAddress ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1a"
    STATUS       current
    DESCRIPTION
        "Represents a SCTP connection address for an IPv4 address, an
        IPv6 address or an ASCII encoded host name and port number.

        The hostname must be encoded in ASCII, as specified in RFC3490
        (Internationalizing Domain Names in Applications) followed by
        a colon ':' (ASCII character 0x3A) and a decimal port number
        in ASCII. The name SHOULD be fully qualified whenever
        possible.

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

        An IPv6 address must be a colon separated format, surrounded
        by square brackets (ASCII characters 0x5B and 0x5D), followed
        by a colon ':' (ASCII character 0x3A) and a decimal port
        number in ASCII.

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

FingerprintType ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
       "Identifies an algorithm type that can be used to uniquely
       reference other data of a potentially arbitrary length.  If a
       hashing algorithm is used, then the algorithm should be
       sufficiently robust enough and it's output long enough that
       hash-collisions should not occur.  Other mechanisms of defining
       fingerprints include other forms of unique identification such
       as serial numbers or concatenated combinations of data such
       that the result is sufficiently unique.

       Objects making use of this TEXTUAL-CONVENTION SHOULD be
       accompanied by another object or objects of type
       FingerprintValue.

       This TEXTUAL-CONVENTION SHOULD NOT be used as a form of
       cryptographic verification. Two matching sets of
       FingerprintType/FingerprintValue should not be considered
       authenticated.  These TEXTUAL-CONVENTIONs are only intended for
       use as a reference to a stored copy of a longer data source and
       the full data sources need to be compared to assure collisions
       have not resulted."
    SYNTAX       OBJECT IDENTIFIER

FingerprintValue ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
       "A Fingerprint value that can be used to uniquely reference
       other data of potentially arbitrary length.

       Objects making use of this TEXTUAL-CONVENTION SHOULD always be
       accompanied by a FingerprintType object that dictates the
       format of values stored in objects of type FingerprintValue.

       This TEXTUAL-CONVENTION SHOULD NOT be used as a form of
       cryptographic verification. Two matching sets of
```

```
       FingerprintType/FingerprintValue should not be considered
       authenticated.  These TEXTUAL-CONVENTIONs are only intended for
       use as a reference to a stored copy of a longer data source and
       the full data sources need to be compared to assure collisions
       have not resulted."
    SYNTAX       OCTET STRING

-- The Fingerprint Identifiers Objects

tlstmFingerprintTypes  OBJECT IDENTIFIER ::= { tlstmObjects 1 }

tlstmMD5         OBJECT-IDENTITY
    STATUS       current
    DESCRIPTION
       "An identifier for the MD5 hashing algorithm to be used with
       FingerprintType objects.  The resulting value to be placed into
       the corresponding FingerprintValue object should be a
       full-length MD5 hash (16 octets)."
    ::= { tlstmFingerprintTypes 1 }

tlstmSHA1         OBJECT-IDENTITY
    STATUS       current
    DESCRIPTION
       "An identifier for the SHA1 hashing algorithm to be used with
       FingerprintType objects.  The resulting value to be placed into
       the corresponding FingerprintValue object should be a
       full-length SHA1 hash (20 octets)."
    ::= { tlstmFingerprintTypes 2 }

tlstmSHA256         OBJECT-IDENTITY
    STATUS       current
    DESCRIPTION
       "An identifier for the SHA256 hashing algorithm to be used with
       FingerprintType objects.  The resulting value to be placed into
       the corresponding FingerprintValue object should be a
       full-length SHA256 hash (32 octets)."
    ::= { tlstmFingerprintTypes 3 }

-- The tlstmSession Group

tlstmSession           OBJECT IDENTIFIER ::= { tlstmObjects 2 }

tlstmSessionOpens  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, whether it succeeded or failed."
    ::= { tlstmSession 1 }

tlstmSessionCloses  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, whether it succeeded or failed."
    ::= { tlstmSession 2 }

tlstmSessionOpenErrors  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."
    ::= { tlstmSession 3 }


tlstmSessionNoAvailableSessions  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."
    ::= { tlstmSession 4 }

tlstmSessionInvalidClientCertificates 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 includes, but is not
        limited to, cryptographic validation failures and lack of a
        suitable mapping row in the tlstmCertToSNTable."
    ::= { tlstmSession 5 }

tlstmSessionInvalidServerCertificates 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 presented server certificate was
        invalid.  Reasons for invalidation includes, but is not
        limited to, cryptographic validation failures and an unexpected
        presented certificate identity."
    ::= { tlstmSession 6 }

tlstmTLSProtectionErrors OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times (D)TLS processing resulted in a message
        being discarded because it failed its integrity test,
        decryption processing or other (D)TLS processing."
    ::= { tlstmSession 7 }

-- Configuration Objects

tlstmConfig          OBJECT IDENTIFIER ::= { tlstmObjects 3 }

-- Certificate mapping

tlstmCertificateMapping    OBJECT IDENTIFIER ::= { tlstmConfig 1 }

tlstmCertToSNCount OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "A count of the number of entries in the
        tlstmCertToSNTable"
    ::= { tlstmCertificateMapping 1 }

tlstmCertToSNTableLastChanged OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of sysUpTime.0 when the tlstmCertToSNTable
        was last modified through any means, or 0 if it has not been
        modified since the command responder was started."
    ::= { tlstmCertificateMapping 2 }

tlstmCertToSNTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF TlstmCertToSNEntry
    MAX-ACCESS  not-accessible
```

```
    STATUS      current
    DESCRIPTION
        "A table listing the 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 SNMPv3 securityName.
        This table does not provide any mechanisms for uploading
        trusted certificates and 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 securityName
        to identify with the remote connection.  This is done by
        considering each active row from this table in prioritized
        order according to its tlstmCertToSNID value.  Each row's
        tlstmCertToSNHashType and tlstmCertToSNHashValue values
        determine whether the row is a match for the incoming
        connection:

            1) If the row's tlstmCertToSNHashType and
               tlstmCertToSNHashValue values identify the presented
               certificate and the contents of the presented
               certificate match a locally cached copy of the
               certificate then consider the row as a successful
               match.

            2) If the row's tlstmCertToSNHashType and
               tlstmCertToSNHashValue values identify a locally held
               copy of a trusted CA certificate and that CA
               certificated was used to validate the presented
               certificate then consider the row as a successful
               match.

        Once a matching row has been found, the tlstmCertToSNMapType
        value can be used to determine how the securityName to
        associate with the session should be determined.  See the
        tlstmCertToSNMapType column's DESCRIPTION for details on
        determining the securityName value.  If it is impossible to
        determine the resulting securityName 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 securityName mapped from a given row is
```

```
        not compatible with the needed requirements of a legal
        securityName (i.e., VACM imposes a 32-octet-maximum length)
        then it must be considered an invalid match and additional
        rows MUST be searched looking for another potential match.

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

        Users are encouraged to make use of certificates with
        subjectAltName fields that can be used as securityNames so
        that a single root CA certificate can allow all child
        certificate's subjectAltName to map directly to a securityName
        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 SNMPv3 securityNames.  Certificates may also be
        mapped to securityNames using the CommonName portion of the
        Subject field but usage of the CommonName field is deprecated.
        Direct mapping from each individual certificate fingerprint to
        a securityName is also possible but requires one entry in the
        table per securityName and requires more management operations
        to completely configure a device."
    ::= { tlstmCertificateMapping 3 }

tlstmCertToSNEntry OBJECT-TYPE
    SYNTAX      TlstmCertToSNEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A row in the tlstmCertToSNTable that specifies a
        mapping for an incoming (D)TLS certificate to a securityName
        to use for a connection."
    INDEX   { tlstmCertToSNID }
    ::= { tlstmCertToSNTable 1 }

TlstmCertToSNEntry ::= SEQUENCE {
    tlstmCertToSNID           Unsigned32,
    tlstmCertToSNHashType     FingerprintType,
    tlstmCertToSNHashValue    FingerprintValue,
    tlstmCertToSNMapType      INTEGER,
    tlstmCertToSNSecurityName SnmpAdminString,
    tlstmCertToSNSANType      INTEGER,
    tlstmCertToSNStorageType  StorageType,
    tlstmCertToSNRowStatus    RowStatus
}
```

```
tlstmCertToSNID OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A unique, prioritized index for a given certificate entry."
    ::= { tlstmCertToSNEntry 1 }

tlstmCertToSNHashType  OBJECT-TYPE
    SYNTAX      FingerprintType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
       "The hash algorithm to use when applying a hash to a X.509
       certificate for purposes of referring to it from the
       tlstmCertToSNHashValue column."
    DEFVAL { tlstmSHA256 }
    ::= { tlstmCertToSNEntry 2 }


tlstmCertToSNHashValue OBJECT-TYPE
    SYNTAX      FingerprintValue
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "A cryptographic hash of a X.509 certificate.  The results of
        a successful matching fingerprint is dictated by the
        tlstmCertToSNMapType column.  A match of the fingerprint MUST
        only be considered successful if both the fingerprint type
        (tlstmCertToSNMapType) and fingerprint value
        (tlstmCertToSNHashValue) columns have been found equal to the
        type and value of the certificate being checked."
    ::= { tlstmCertToSNEntry 3 }

tlstmCertToSNMapType OBJECT-TYPE
    SYNTAX      INTEGER { specified(1), bySubjectAltName(2), byCN(3) }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The mapping type used to obtain the securityName from the
        certificate.  The possible values of use and their usage
        methods are defined as follows:

        specified(1): The securityName that should be used to
                      associate with the session is directly
                      specified in the tlstmCertToSNecurityName column
                      from this table.
```

```
        bySubjectAltName(2):
                     The securityName that should be used to
                     associate with the session should be taken from
                     the subjectAltName(s) portion of the client's
                     X.509 certificate.  The subjectAltName used MUST
                     be the first encountered subjectAltName type
                     indicated by the tlstmCertToSNSANType column.
                     If no subjectAltName of the given type is found
                     within the certificate then additional rows in
                     the tlstmCertToSNTable must be searched.

        byCN(3):     The securityName that should be used to
                     associate with the session should be taken from
                     the CommonName portion of the Subject field from
                     the client's presented X.509 certificate.

        If the resulting mapped value from the subjectAltName
        component is not compatible with the needed requirements of a
        legal securityName (i.e., VACM imposes a 32-octet-maximum
        length) then the next appropriate subjectAltName of the
        correct type should be used if available.

        If this object is of type bySubjectAltName(2) and no
        subjectAltName value can be found that meets the requirements
        of this object then any additional rows in the
        tlstmCertToSNTable must be searched.

        If this object is of type byCN(3) and the CommonName value
        does not meet the requirements of this object then any
        additional rows in the tlstmCertToSNTable must be searched."

    DEFVAL { specified }
    ::= { tlstmCertToSNEntry 4 }

tlstmCertToSNSecurityName OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE(0..32))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The securityName that the session should use if the
        tlstmCertToSNMapType is set to specified(1), otherwise the
        value in this column should be ignored.  If
        tlstmCertToSNMapType is set to specifed(1) and this column
        contains a zero-length string (which is not a legal
        securityName value) this row is effectively disabled and the
        match will not be considered successful and other rows in the
        table will need to be searched."
    DEFVAL { "" }
```

```
    ::= { tlstmCertToSNEntry 5 }

tlstmCertToSNSANType OBJECT-TYPE
    SYNTAX      INTEGER { any(1), rfc822Name(2), dNSName(3),
                          ipAddress(4), otherName(5) }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "Specifies the subjectAltName type that may be used to extract
        the securityName from.

        The any(1) value indicates the (D)TLS server should use the
        first value found for any of the following subjectAltName
        value types for the securityName: rfc822Name, dNSName, and
        ipAddress.

        When multiple types for a given subjectAltName type are
        encountered within a certificate the first legally usable
        value is the one selected.

        Values for type ipAddress(4) are converted to a valid
        securityName by:

            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.

        Values for type otherName(5) are converted to a valid
        securityName by using only the decoded value portion of the
        OthenName sequence.  i.e. the OBJECT IDENTIFIER portion of the
        OtherName sequence is not included as part of the resulting
        securityName."
    DEFVAL      { any }
    ::= { tlstmCertToSNEntry 6 }

tlstmCertToSNStorageType 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 }
    ::= { tlstmCertToSNEntry 7 }
```

```
tlstmCertToSNRowStatus 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 tlstmCertToSNHashType,
        tlstmCertToSNHashValue, tlstmCertToSNMapType,
        tlstmCertToSNSecurityName, and tlstmCertToSNSANType columns
        have been set.

        The following objects may not be modified while the
        value of this object is active(1):
            - tlstmCertToSNHashType
            - tlstmCertToSNHashValue
            - tlstmCertToSNMapType
            - tlstmCertToSNSecurityName
            - tlstmCertToSNSANType
        An attempt to set these objects while the value of
        tlstmParamsRowStatus is active(1) will result in
        an inconsistentValue error."
    ::= { tlstmCertToSNEntry 8 }

-- Maps securityNames 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 extends the SNMP-TARGET-MIB's
        snmpTargetParamsTable with an additional (D)TLS client-side
        certificate fingerprint identifier to use when establishing
        new (D)TLS connections."
    ::= { tlstmCertificateMapping 6 }

tlstmParamsEntry OBJECT-TYPE
    SYNTAX      TlstmParamsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A conceptual row containing a locally held certificate's hash
        type and hash value 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 usable
        certificate (such as might happen when the certificate's
        expiration date has been reached)."
    INDEX    { IMPLIED snmpTargetParamsName }
    ::= { tlstmParamsTable 1 }

TlstmParamsEntry ::= SEQUENCE {
    tlstmParamsClientHashType  FingerprintType,
    tlstmParamsClientHashValue FingerprintValue,
    tlstmParamsStorageType     StorageType,
    tlstmParamsRowStatus       RowStatus
}

tlstmParamsClientHashType  OBJECT-TYPE
    SYNTAX      FingerprintType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
       "The hash algorithm type for the hash stored in the
       tlstmParamsClientHash column to identify a locally-held X.509
```

```
       certificate that should be used when initiating a (D)TLS
       connection as a (D)TLS client."
    DEFVAL { tlstmSHA256 }
    ::= { tlstmParamsEntry 1 }

tlstmParamsClientHashValue OBJECT-TYPE
    SYNTAX      FingerprintValue
    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 2 }

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 3 }


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 tlstmParamsClientHashType and
        tlstmParamsClientHashValue columns have been set.

        The following objects may not be modified while the
```

```
        value of this object is active(1):
            - tlstmParamsClientHashType
            - tlstmParamsClientHashValue
        An attempt to set these objects while the value of
        tlstmParamsRowStatus is active(1) will result in
        an inconsistentValue error.

        If this row is deleted it has no effect on the corresponding
        row in the targetParamsTable.

        If the corresponding row in the targetParamsTable is deleted
        then this row must be automatically removed."
    ::= { tlstmParamsEntry 4 }



-- Lists expected certificate fingerprints to be presented by a DTLS
-- server

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 extends the SNMP-TARGET-MIB's snmpTargetAddrTable
        with an expected (D)TLS server-side certificate identifier to
        expect when establishing a new (D)TLS connections.  If a
        matching row in this table exists and the row is active then a
        local copy of the certificate matching the fingerprint
        identifier should be compared against the certificate being
```

```
        presented by the server.  If the certificate presented by the
        server does not match the locally held copy then the
        connection MUST NOT be established.  If no matching row exists
        in this table then the connection SHOULD still proceed if
        another certificate validation path algorithm (e.g. RFC5280)
        can be followed to a configured trust anchor. "
    ::= { tlstmCertificateMapping 9 }

tlstmAddrEntry OBJECT-TYPE
    SYNTAX      TlstmAddrEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A conceptual row containing a copy of a locally held
        certificate's hash type and hash value 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
        followed to a configured trust anchor."
    INDEX    { IMPLIED snmpTargetAddrName }
    ::= { tlstmAddrTable 1 }

TlstmAddrEntry ::= SEQUENCE {
    tlstmAddrServerHashType  FingerprintType,
    tlstmAddrServerHashValue FingerprintValue,
    tlstmAddrStorageType     StorageType,
    tlstmAddrRowStatus       RowStatus
}

tlstmAddrServerHashType  OBJECT-TYPE
    SYNTAX      FingerprintType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
       "The hash algorithm type for the hash stored in the
       tlstmAddrServerHash column to identify a local copy of the
       public X.509 certificate presented by the TLS server."
    DEFVAL { tlstmSHA256 }
    ::= { tlstmAddrEntry 1 }

tlstmAddrServerHashValue OBJECT-TYPE
    SYNTAX      FingerprintValue
```

```
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "A cryptographic hash of a public X.509 certificate.  This
        object should store the hash of a local copy of the public
        X.509 certificate that the remote server should present during
        the (D)TLS connection setup.  The presented certificate and
        the locally held copy, referred to by this hash value, MUST
        match exactly or the connection MUST NOT be established."
    ::= { 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 tlstmAddrServerHashType, and
        tlstmAddrServerHashValue columns have been set.

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

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

        If this row is deleted it has no effect on the corresponding
        row in the targetAddrTable.

        If the corresponding row in the targetAddrTable is deleted
        then this row must be automatically removed."
    ::= { tlstmAddrEntry 4 }


-- ************************************************
-- tlstmMIB - 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 }
    ::= { tlstmCompliances 1 }

-- ************************************************
-- Units of conformance
-- ************************************************
tlstmStatsGroup OBJECT-GROUP
    OBJECTS {
        tlstmSessionOpens,
        tlstmSessionCloses,
        tlstmSessionOpenErrors,
        tlstmSessionNoAvailableSessions,
        tlstmSessionInvalidClientCertificates,
        tlstmSessionInvalidServerCertificates,
        tlstmTLSProtectionErrors
```

```
    }
    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 {
        tlstmCertToSNCount,
        tlstmCertToSNTableLastChanged,
        tlstmCertToSNHashType,
        tlstmCertToSNHashValue,
        tlstmCertToSNMapType,
        tlstmCertToSNSecurityName,
        tlstmCertToSNSANType,
        tlstmCertToSNStorageType,
        tlstmCertToSNRowStatus
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects for maintaining
        incoming connection certificate mappings to
        securityNames of an SNMP engine which implements the
        SNMP TLS Transport Model."
    ::= { tlstmGroups 2 }

tlstmOutgoingGroup OBJECT-GROUP
    OBJECTS {
        tlstmParamsCount,
        tlstmParamsTableLastChanged,
        tlstmParamsClientHashType,
        tlstmParamsClientHashValue,
        tlstmParamsStorageType,
        tlstmParamsRowStatus,
        tlstmAddrCount,
        tlstmAddrTableLastChanged,
        tlstmAddrServerHashType,
        tlstmAddrServerHashValue,
        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 }

END
```

8. Operational Considerations

This section discusses various operational aspects of the solution

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 and this information is created and destroyed as sessions are opened and closed. Because of the connectionless nature of UDP, a "broken" session, one side up one side down, could 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 dealing with "broken" sessions. Implementations SHOULD support the session resumption feature of TLS.

To simplify session management it is RECOMMENDED that implementations utilize two separate ports, one for Notification sessions and one for Command sessions. If this implementation recommendation is followed, (D)TLS clients will always send REQUEST messages and (D)TLS servers will always send RESPONSE messages. With this assertion, implementations may be able to simplify "broken" session handling, session resumption, and other aspects of session management such as guaranteeing that Request- Response pairs use the same session.

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. However, the receiving SNMP engine (Server) does not know which (D)TLS certificate to offer to the Client so that the tmSecurityName identity-authentication will be successful. The best solution would be to maintain a one-to-one mapping between certificates and incoming ports for notification

receivers, although other implementation dependent mechanisms may be used instead. This can be handled at the Notification Originator by configuring the snmpTargetAddrTable (snmpTargetAddrTDomain and snmpTargetAddrTAddress) and then 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 TRAPS and INFORMS.

8.3. contextEngineID Discovery

Because most Command Responders have contextEngineIDs that are identical to the USM securityEngineID, the USM provides Command Generators with the ability to discover a default contextEngineID to use. Because the TLS Transport Model does not make use of a discoverable securityEngineID like the USM does, 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 contextEngineID discovery mechanism. A recommended discovery solution is documented in [RFC5343].

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 and therefore it is RECOMMENDED that implementers also support this cookie exchange.

9.1. Certificates, Authentication, and Authorization

Implementations are responsible for providing a security certificate configuration installation . Implementations SHOULD support certificate revocation lists and expiration of certificates or other access control mechanisms.

(D)TLS provides for both authentication of the identity of the (D)TLS server and authentication of the identity of 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.

Furthermore, 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-positive 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 tlstmCertToSNTable object must be followed exactly. Specifically, it is important that if a row matching a certificate or a certificate's issuer is found but the translation to a securityName using the row fails that the lookup process stops and no further rows are consulted. It is also important that the rows of the table be search in order starting with the row containing the lowest numbered tlstmCertToSNID value.

9.2. Use with SNMPv1/SNMPv2c Messages

The SNMPv1 and SNMPv2c message processing described in RFC3484 (BCP 74) [RFC3584] always selects the SNMPv1(1) Security Model for an SNMPv1 message, or the SNMPv2c(2) Security Model for an SNMPv2c message. When running SNMPv1/SNMPv2c over a secure transport like the TLS Transport Model, the securityName and securityLevel used for access control decisions are then derived from the community string, not 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:

- 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.

- 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.

- The tlstmCertToSNTable is used to specify the mapping of incoming X.509 certificates to SNMPv3 securityNames. 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:

- 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 in the range 1..1023 in the http://www.iana.org/assignments/port-numbers registry which will be the default port for SNMP command messages over a TLS Transport Model as defined in this document,

2. a TCP port number in the range 1..1023 in the http://www.iana.org/assignments/port-numbers registry which will be the default port for SNMP notification messages over a TLS Transport Model as defined in this document,

3. a UDP port number in the range 1..1023 in the http://www.iana.org/assignments/port-numbers registry which will be the default port for SNMP command messages over a DTLS/UDP connection as defined in this document,

4. a UDP port number in the range 1..1023 in the http://www.iana.org/assignments/port-numbers registry which will be the default port for SNMP notification messages over a DTLS/UDP connection as defined in this document,

5. a SCTP port number in the range 1..1023 in the http://www.iana.org/assignments/port-numbers registry which will be the default port for SNMP command messages over a DTLS/SCTP connection as defined in this document,

6. a SCTP port number in the range 1..1023 in the http://www.iana.org/assignments/port-numbers registry which will be the default port for SNMP notification messages over a DTLS/SCTP connection as defined in this document,

7. an SMI number under snmpDomains for the snmpTLSDomain object identifier,

8. an SMI number under snmpDomains for the snmpDTLSUDPDomain object identifier,

9. an SMI number under snmpDomains for the snmpDTLSSCTPDomain object identifier,

10. a SMI number under snmpModules, for the MIB module in this document,

11. "tls" as the corresponding prefix for the snmpTLSDomain in the SNMP Transport Model registry,

12. "dudp" as the corresponding prefix for the snmpDTLSUDPDomain in the SNMP Transport Model registry,

13. "dsct" as the corresponding prefix for the snmpDTLSSCTPDomain in the SNMP Transport Model registry;

14. 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 [I-D.ietf-isms-secshell].

This document was reviewed by the following people who helped provide useful comments: David Harrington, Alan Luchuk, Ray Purvis.

This work was supported in part by the United States Department of Defense. Large portions of this document are based on work by General Dynamics C4 Systems and the following individuals: Brian

Baril, Kim Bryant, Dana Deluca, Dan Hanson, Tim Huemiller, John Holzhauer, Colin Hoogeboom, Dave Kornbau, Chris Knaian, Dan Knaul, Charles Limoges, Steve Moccaldi, Gerardo Orlando, and Brandon Yip.

12. References

12.1. Normative References

[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.

[I-D.ietf-isms-transport-security-model] Harington, D., "Transport Security Model for SNMP".

[I-D.ietf-isms-tmsm] Harington, D. and J. Schoenwaelder, "Transport Subsystem for the Simple Network Management Protocol (SNMP)".

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.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, December 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[I-D.ietf-isms-secshell] Harington, D. and J. Salowey, "Secure Shell Transport Model for SNMP".

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

[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)".

[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] Rivest, R., Shamir, A., and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems".

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:

- The session can be confidential. Symmetric cryptography is used for data encryption (e.g., AES [AES], DES [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.

- 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.

- 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 three basic properties:

- 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.

- 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.

- 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 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, which was first published in 1988 as part of the X.500 Directory recommendations, defines a standard certificate format [x509] which is a certificate which binds a subject (principal) to a public key value. This was later further documented in [RFC5280].

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

tbsCertificate: The 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 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, a 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 decrypted information with a digest of the tbsCertificate field. If they match, then the subject in the tbsCertificate field is authenticated.

Appendix C. Target and Notificaton 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 XXX corresponding to the TSM [I-D.ietf-isms-transport-security-model]. An example vacmSecurityToGroupTable row might be filled out as follows (using a single SNMP SET request):

Note to RFC editor: replace XXX in the previous paragraph above with the actual IANA-assigned number for the TSM security model and remove this note.

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

Note to RFC editor: replace XXX in the vacmSecurityModel line above with the actual IANA-assigned number for the TSM security model and remove this note.

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.

For Notification Generator's performing authorization checks, the server's certificate must be verified against the expected certificate before proceeding to send the notification. The securityName be set by the SNMP-TARGET-MIB's snmpTargetParamsSecurityName column or other configuration mechanism and the certificate to use would be taken from the appropriate entry in the tlstmParamsTable. The tlstmParamsTable augments the SNMP-TARGET-MIB's snmpTargetParamsTable with client-side certificate information.

For Command Responder applications, the vacmSecurityName "blueberry" value is a value that needs to come from an incoming (D)TLS session. The mapping from a recevied (D)TLS client certificate to a securityName is done with the tlstmCertToSNTable. The certificates must be loaded into the device so that a tlstmCertToSNEntry may refer to it. As an example, consider the following entry which will provide a mapping from a X.509's hash fingerprint directly to the "blueberry" securityName:

```
  tlstmCertToSNID           = 1       (chosen by ordering preference)
  tlstmCertToSNHashType     = tlstmSHA256
  tlstmCertToSNHashValue    = (appropriate sha256 fingerprint)
  tlstmCertToSNMapType      = specified(1)
  tlstmCertToSNSecurityName = "blueberry"
  tlstmCertToSNStorageType  = 3 (nonVolatile)
  tlstmCertToSNRowStatus    = 4 (createAndGo)
```

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

```
tlstmCertToSNID          = 1       (chosen by ordering preference)
tlstmCertToSNHashType    = tlstmSHA256
tlstmCertToSNHashValue   = (appropriate sha256 fingerprint)
tlstmCertToSNMapType     = bySubjectAltName(2)
tlstmCertToSNSANType     = 1 (any)
tlstmCertToSNStorageType = 3 (nonVolatile)
tlstmCertToSNRowStatus   = 4 (createAndGo)
```

The above entry indicates the subjectAltName field for certificates created by an Issuing certificate with a corresponding hash type and value will be trusted to always produce common names that are directly 1 to 1 mappable into SNMPv3 securityNames. This type of configuration should only be used when the certificate authorities naming conventions are carefully controlled.

For 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" and the certificate was signed by a certificate matching the tlstmCertToSNHashType and tlstmCertToSNHashValue values above and the CA's certificate was properly installed on the device then the string "blueberry" would be used as the securityName for the session.

Author's Address

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

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