ISMS
Internet-Draft
Intended status: Informational
Expires: May 7, 2009

W. Hardaker
Sparta, Inc.

November 3, 2008

Datagram Transport Layer Security Transport Model for SNMP
draft-hardaker-isms-dtls-tm-01.txt

Status of this Memo

Abstract

This document describes a Transport Model for the Simple Network Management Protocol (SNMP), that uses the Datagram Transport Layer Security (DTLS) protocol. The DTLS protocol provides authentication and privacy services for SNMP applications. This document describes how the DTLS Transport Model (DTLSTM) 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, operate in environments where a connectionless (UDP) transport is preferred, and integrates well into existing public keying infrastructures.

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

Table of Contents

1. Introduction

It is important to understand the SNMPv3 architecture [RFC3411], as 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].

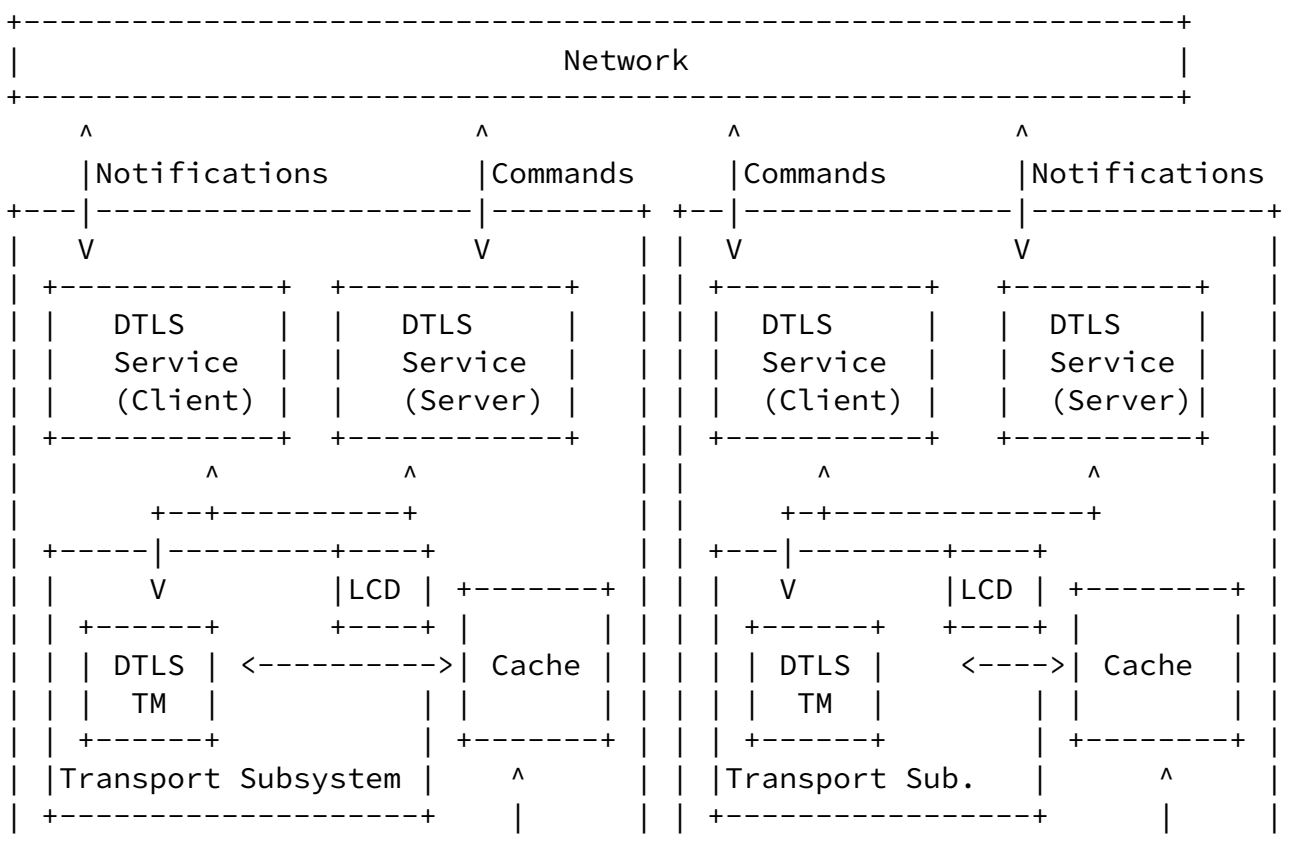
This document describes a Transport Model that makes use of the Datagram Transport Layer Security (DTLS) Protocol [RFC4347], the datagram variant of the existing and commonly deployed Transport Layer Security (TLS) protocol [RFC4346], within a transport subsystem [I-D.ietf-isms-tmsm]. The Transport Model in this document is referred to as the Datagram Transport Layer Security Transport Model (DTLSTM). DTLS takes advantage of the X.509 public keying infrastructure [X509]. This transport model is designed to meet the security and operational needs of network administrators, operate in environments where a connectionless (UDP) transport is preferred, and integrate well into existing public keying infrastructures.

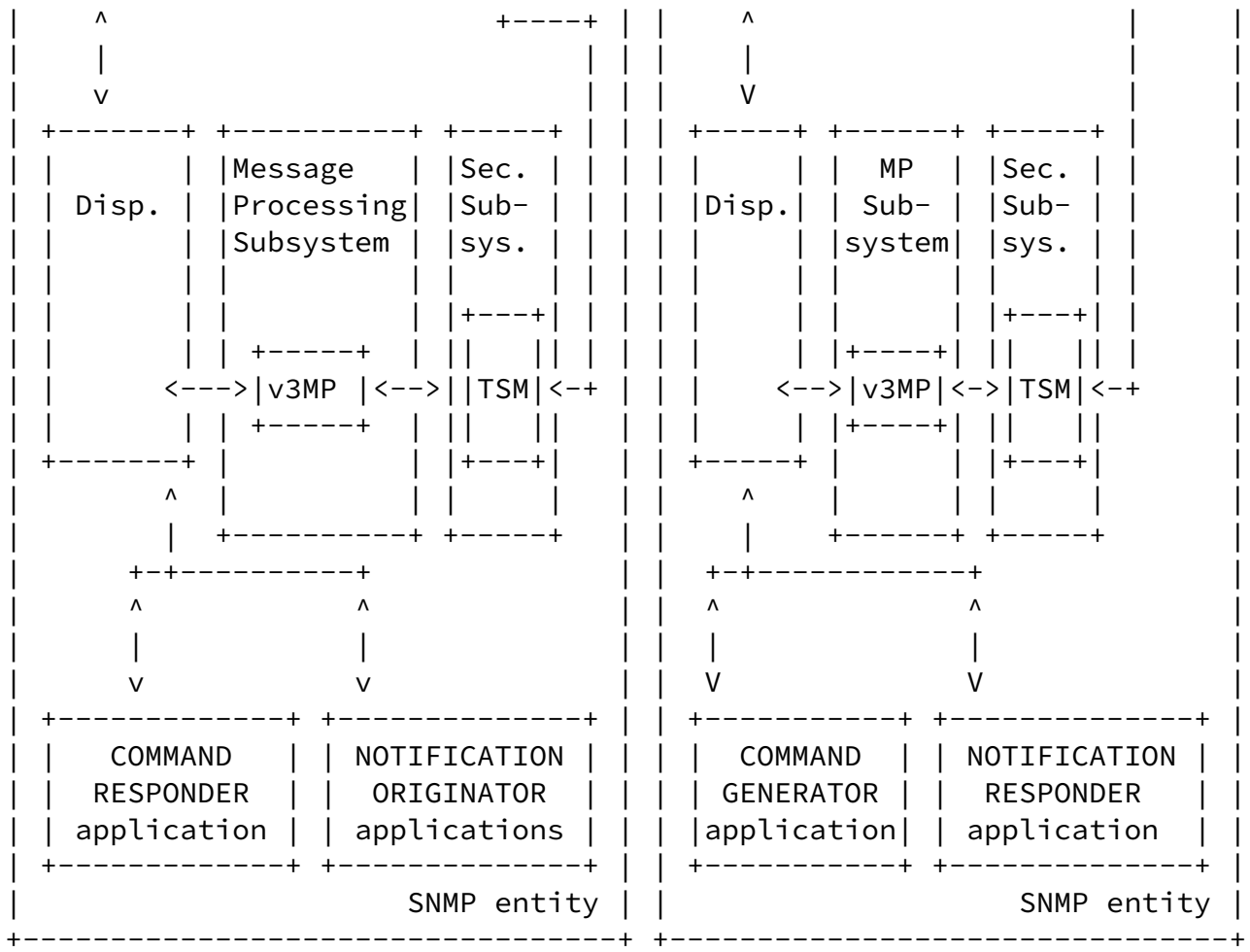
Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information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

58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

This document also defines a portion of the Management Information Base (MIB) for use with network management protocols in IP based networks. In particular it defines objects for monitoring and managing the DTLS Transport Model for SNMP.

The diagram shown below gives a conceptual overview of two SNMP entities communicating using the DTLS 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. Requirements Terminology

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

1.2. 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 that use different variations of the same terminology. This is consistent with the IESG decision to not require the SNMPv3 terminology be modified to match the usage of other non-SNMP specifications when SNMPv3 was advanced to Full Standard.

In particular, where distinction is required the application names of "Command Generator", "Command Responder", "Notification Originator", "Notification Receiver", and "Proxy Forwarder" are used. See the "SNMP Applications" in [RFC3413] for further information.

Authentication in this document typically refers to source authentication or peer identity authentication performed in the transport subsystem.

Throughout this document, the terms "client" and "server" are used to refer to the two ends of the DTLS session. The client actively opens the DTLS session, and the server passively listens for the incoming DTLS session. Any SNMP entity may act as client or as server.

While security protocols frequently refer to a "user", the term used in RFC3411 [RFC3411] and in this document 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 DTLS Transport Models that permits the transmission of one or more SNMP messages within the lifetime of the session.

2. The Datagram Transport Layer Security Protocol

The DTLS protocol is a datagram-compatible variant of the commonly

used Transport Layer Security (TLS) protocol. DTLS provides authentication, data message integrity, and privacy at the transport layer. (See [RFC4347])

The primary goals of the DTLS Transport Model are to provide privacy, source authentication and data integrity between two communicating SNMP entities. The DTLS protocol is composed of two layers: the DTLS Record Protocol and the DTLS Handshake Protocol. The following

sections provide an overview of these two layers. Please refer to [RFC4347] for a complete description of the protocol.

2.1. The DTLS Record Protocol

At the lowest layer, layered on top of the transport protocol (UDP) is the DTLS Record Protocol.

The DTLS 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 DTLS 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. DTLS uses explicit sequence numbers, integrity checks, and a sliding window to protect against replay of messages within a session.

DTLS 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.

2.2. The DTLS Handshake Protocol

The DTLS Record Protocol is used for encapsulation of various higher-level protocols. One such encapsulated protocol, the DTLS Handshake Protocol, allows server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the

application protocol transmits or receives its first octet of data.

Only the DTLS client can initiate the handshake protocol. The DTLS Handshake Protocol provides security that has three basic properties:

- The peer's identity can be authenticated using asymmetric, or 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.

DTLS 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. The DTLS Transport Model SHOULD always use authentication of both the server and the client. At a minimum the DTLS Transport MUST support authentication of the Command Generator principals to guarantee the authenticity of the securityName (a parameter used to pass the authenticated identity name from the transport model to security model for even later use by the access control subsystem. See Section 4.5.3.4). The DTLS Transport SHOULD support the message encryption to protect sensitive data from eavesdropping attacks. See [I-D.hodges-server-ident-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.

- 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.

3. How the DTLSTM fits into the Transport Subsystem

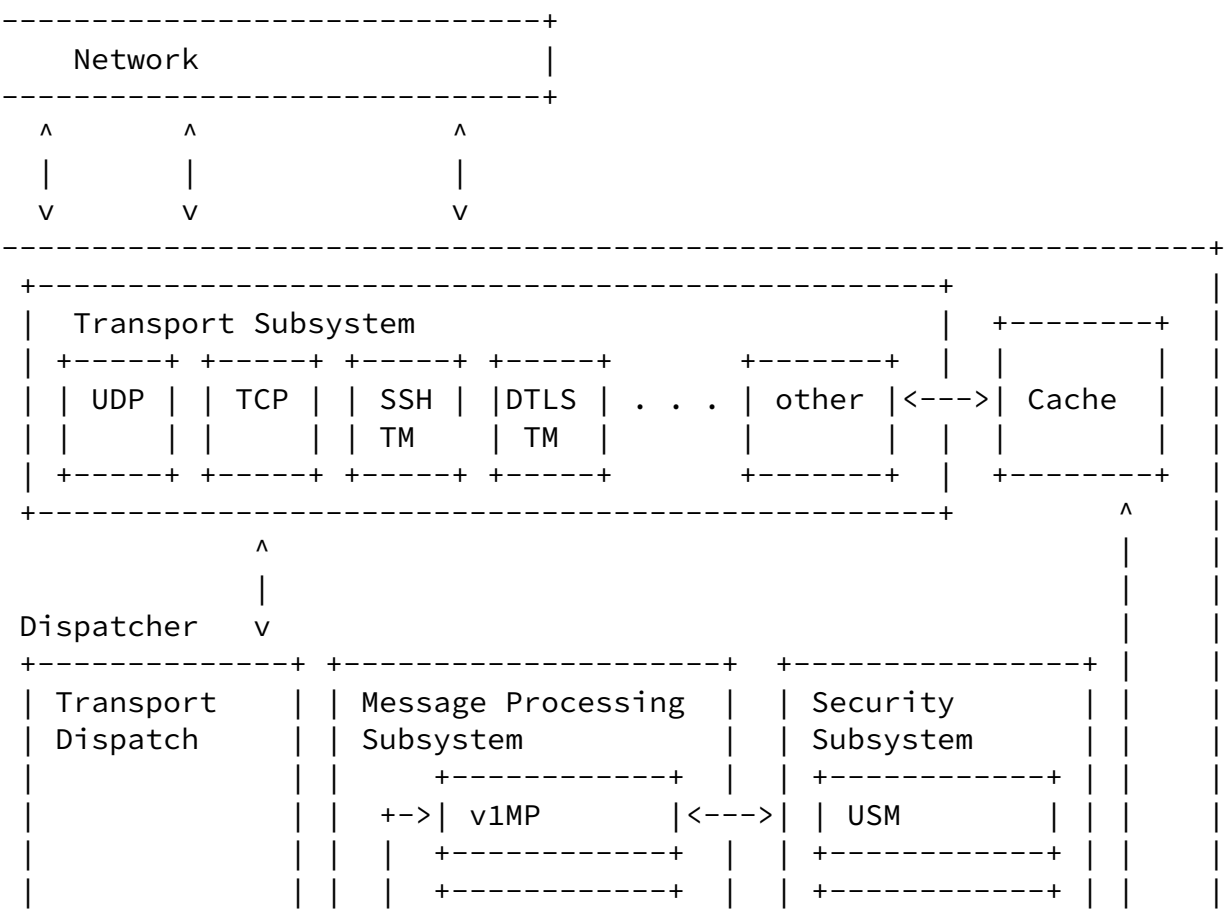
A transport model is a component of the Transport Subsystem. The DTLS Transport Model thus fits between the underlying DTLS transport layer and the message Dispatcher [RFC3411] component of the SNMP engine and the Transport Subsystem [I-D.ietf-isms-tmsm].

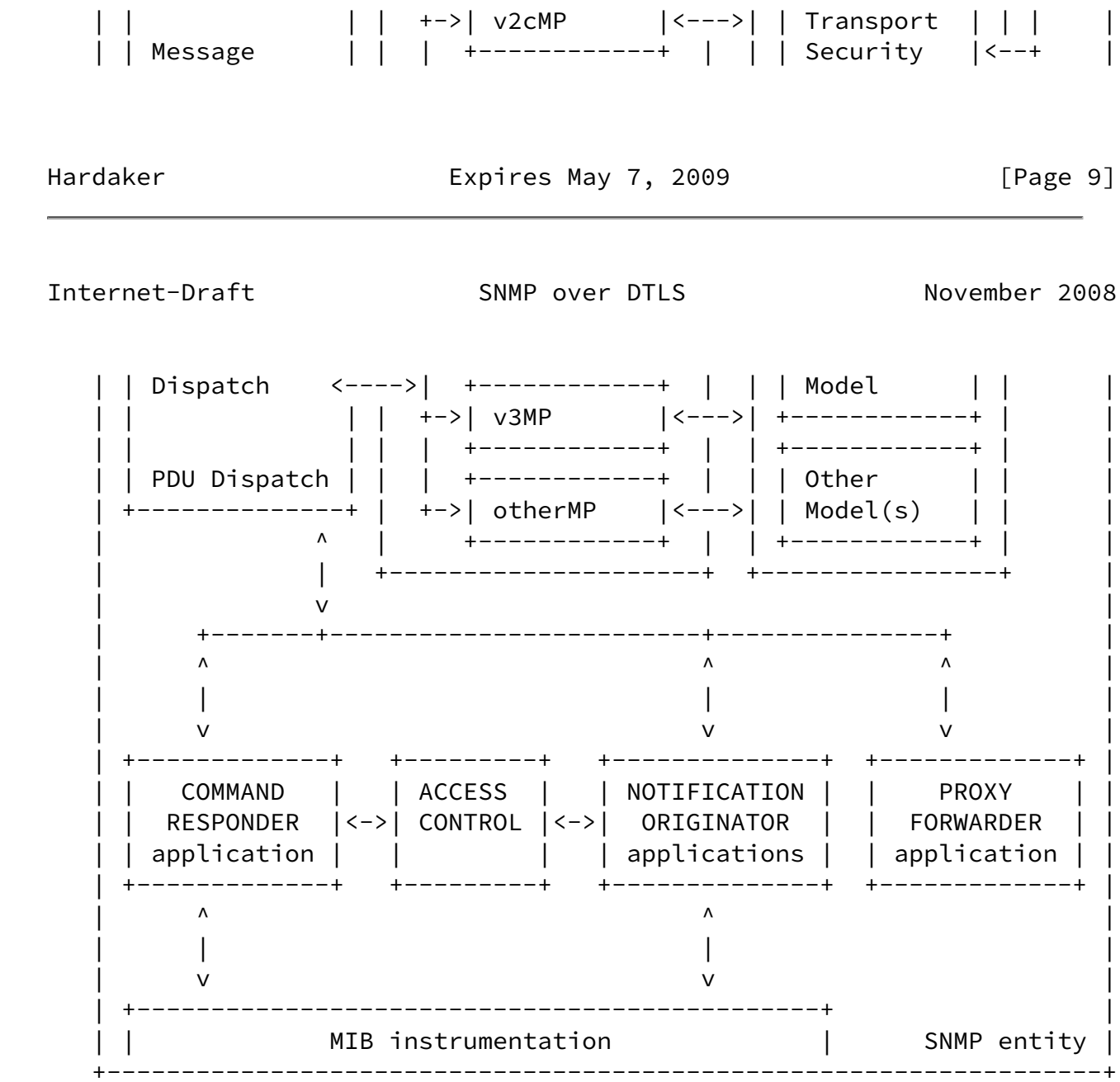
The DTLS Transport Model will establish a session between itself and the DTLS Transport Model of another SNMP engine. The sending transport model passes unprotected messages from the dispatcher to DTLS to be protected, and the receiving transport model accepts decrypted and authenticated/integrity-checked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

After a DTLS 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 DTLSTM implementation engine MAY choose to close a DTLS session to conserve resources.

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

The diagram below depicts where the DTLS 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 DTLS 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.

   DTLS 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 a principal with appropriate authorizations.

The DTLSTM provides for authentication of the Principal Command Generator and Notification Generator and for authentication of the Command Responder, 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 DTLS client and the DTLS 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.

DTLS provides replay protection with a MAC that includes a sequence number. DTLS uses a sliding window protocol with the sequence number for replay protection, see [RFC4347]. The technique used is the same 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 as a matter of local policy.

   Symmetric cryptography (e.g., AES [AES], DES [DES] etc.) can be used by DTLS 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 DTLS 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 security protocols are susceptible to a variety of denial of service attacks. Two

attacks are of particular concern:

- An attacker can consume excessive resources on the server by transmitting a series of handshake initiation requests, causing the server to allocate state and potentially to perform expensive cryptographic operations.

- An attacker can use the server as an amplifier by sending session initiation messages with a forged source of the victim. The server then sends its next message (in DTLS, a Certificate message, which can be quite large) to the victim machine, thus flooding it.

In order to counter both of these attacks, DTLS borrows the stateless cookie technique used by Photuris [RFC2522] and IKEv2 [RFC4306]. When the client sends its ClientHello message to the server, the server MAY respond with a HelloVerifyRequest message. This message contains a stateless cookie generated using the technique of [RFC2522]. The client MUST retransmit the ClientHello with the cookie added. The server then verifies the cookie and proceeds with the handshake only if it is valid. This mechanism forces the attacker/client to be able to receive the cookie, which makes DoS attacks with spoofed IP addresses difficult. This mechanism does not provide any defense against

denial of service attacks mounted from valid IP addresses.

Implementations are not required to perform the stateless cookie exchange for every DTLS handshakes but in environments where amplification could be an issue 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 DTLS Transport Model determines from DTLS the identity of the authenticated principal, and the type and address associated with an incoming message, and the DTLS Transport Model provides this information to DTLS 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 DTLS Transport Model MUST ensure that the DTLS 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 DTLS to only allow cipher_suites that provide both authentication and privacy to guarantee this assertion.

If a suitable interface between the DTLS Transport Model and the DTLS 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 DTLS sessions. This is the only way to

ensured proper selection of a session ID for an incoming DTLS message.

The authentication, integrity and privacy algorithms used by the DTLS Protocol [RFC4347] may vary over time as the science of cryptography continues to evolve and the development of DTLS 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. DTLS Sessions

DTLS sessions are opened by the DTLS Transport Model during the elements of procedure for an outgoing SNMP message. Since the sender of a message initiates the creation of a DTLS session if needed, the DTLS session will already exist for an incoming message.

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

DTLS sessions are uniquely identified within the DTLS 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.

3.2. Security Parameter Passing

For the DTLS server-side, DTLS-specific security parameters (i.e., cipher_suites, common name of X.509 certificate, IP address and port) are translated by the DTLS Transport Model into security parameters for the DTLS Transport Model and security model (i.e., securityLevel, securityName, transportDomain, transportAddress). The transport-related and DTLS-security-related information, including the authenticated identity, are stored in a cache referenced by tmStateReference.

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

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

3.3. Notifications and Proxy

DTLS sessions may be initiated by DTLS 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 domain and transport address 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 DTLS client-side certificate to use for the connection.

When configuring a DTLS target, the snmpTargetAddrTDomain and snmpTargetAddrTAddress parameters in snmpTargetAddrTable should be set to the snmpDTLSDomain object and an appropriate snmpDTLSAddress 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 dtlstmParamsSubject parameter of the dtlstmParamsTable should be set

to the Subject of the locally held certificate to be used. 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 DTLS Transport Model defined by this document.

4.1. Certificates

DTLS makes use of X.509 certificates for authentication of both sides of the transport. This section discusses the use of certificates in DTLS and the its effects on SNMP over DTLS.

4.1.1. The Certificate Infrastructure

Users of a public key SHALL 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 extensions.

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, which uses the X.509 key management infrastructure, is initialized with a number of root certificates which contain the public keys of a number of trusted CAs. When a system receives a X.509 certificate, signed by one of those CAs, that has to be verified, it first decrypts the signatureValue field by using the public key of the corresponding trusted CA. Then it compares the decrypted information with the tbsCertificate field. If they match, then the subject in the tbsCertificate field is authenticated.

4.1.2. Provisioning for the Certificate

Authentication using DTLS 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 MAY also be provisioned with X.509 certificate revocation mechanism which will be used to verify that a certificate has not been revoked.

The authenticated tmSecurityName of the principal is looked up using the dtlstmCertificateToSNTable. This table maps the certificates issuer's distinguished name to a directly specified tmSecurityName or it specifies that the CommonName field of the certificate's Subject should be used as the tmSecurityName. 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 MAY choose to discard any connections for which no dtlstmCertificateToSNTable mapping exists for the issuer to avoid the computational resources associated with a certificate verification check since the verified certificate would be unusable anyway.

The typical enterprise configuration will map the "CommonName" component of the Subject field in the tbsCertificate to the DTLSSM specific tmSecurityName. Thus, the authenticated identity can be verified by the DTLS Transport Model by extracting the CommonName from the Subject of the peer 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 dtlstmCertificateToSNTable.

An example mapping setup can be found in Appendix A

This tmSecurityName may be later translated from a DTLSSM 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.

4.2. Messages

As stated in RFC4347, each DTLS message must fit within a single datagram. The DTLSTM MUST prohibit SNMP messages from being set that exceed the MTU size. The DTLSTM implementation MUST return an error when the MTU size would be exceeded and the message won't be sent.

For Ethernet the MTU size is 1500 and thus the maximum allowable SNMP message that can be sent over DTLSTM after UDP/IP/DTLS overhead is taken into account will be 1455 for IPv6 (MTU:1500 - IPv6:40 - UDP:8 - DTLS:13) and 1475 for IPv4 (MTU:1500 - IPv4:20 - UDP:8 - DTLS:13). From this integrity and encryption overhead also needs to be subtracted, which are integrity and encryption algorithm specific.

4.3. SNMP Services

This section describes the services provided by the DTLS 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 DTLS 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.

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

outgoingMessage: The outgoing message to send to DTLS 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 DTLS Transport Model processes the received message from the network using the DTLS service and then passes it to the Dispatcher using the following ASI:


```
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.

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

incomingMessage: The whole SNMP message stripped of all DTLS protection data.

incomingMessageLength: The length of the SNMP message after being

processed by DTLS.

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

4.4. DTLS Services

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

4.4.1. Services for Establishing a Session

The DTLS Transport Model provides the following ASI to describe the data passed between the Transport Model and the DTLS 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  dtlsSessionID           -- Session identifier for DTLS
)
```

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 DTLS.

destTransportDomain: The transport domain for the associated destTransportAddress. The DTLS Transport Model uses this parameter to determine the transport type of the associated

destTransportAddress.

destTransportAddress: The transport address of the destination DTLS Transport Model in a format specified by the SnmpDTLSAddress 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 DTLS 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.

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

Upon completion of the process the DTLS Transport Model returns status information and, if the process was successful the

dtlsSessionID and other implementation-dependent data from DTLS 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. DTLS Services for an Incoming Message

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

```
statusInformation =           -- errorIndication or success
dtlsRead(
IN   dtlsSessionID            -- Session identifier for DTLS
IN   wholeDtlsMsg             -- as received on the wire
IN   wholeDtlsMsgLength       -- length as received on the wire
OUT  incomingMessage          -- the whole SNMP message from DTLS
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 DTLS.

dtlsSessionID: An implementation-dependent session identifier to reference the specific DTLS session. How the DTLS session ID is obtained for each message is implementation-dependent. As an implementation hint, the DTLS Transport Model can examine incoming messages to determine the source IP address and port number and use these values to look up the local DTLS session ID 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 DTLS privacy and integrity data.

incomingMessageLength: The length of the SNMP message stripped of all DTLS privacy and integrity data.

4.4.3. DTLS Services for an Outgoing Message

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

```
statusInformation =           -- errorIndication or success
dtlsWrite(
IN   dtlsSessionID            -- Session identifier for DTLS
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 DTLS.

dtlsSessionID: An implementation-dependent session identifier to reference the specific DTLS session that the message should be sent using.

outgoingMessage: The outgoing message to send to DTLS 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.

The RFC3411 architecture uses caches to maintain the short-term message state, and uses references in the ASIs to pass this information between subsystems.

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

To simplify the elements of procedure, the release of state information is not always explicitly specified. As a general rule, if state information is available when a message being processed gets

discarded, the state related to that message SHOULD also be discarded. If state information is available when a relationship between engines is severed, such as the closing of a transport session, the state information for that relationship SHOULD also be discarded.

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

4.5.1. securityStateReference

The securityStateReference parameter is defined in RFC3411. Its primary purpose is to provide a mapping between a request and the corresponding response. This cache is not accessible to Transport Models, and an entry is typically only retained for the lifetime of a request-response pair of messages.

4.5.2. tmStateReference

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

The tmStateReference cache will typically remain valid for the duration of the transport session, and hence may be used for several messages.

Since this cache is only used within an implementation, and not on-the-wire, the precise contents and format are implementation-dependent. However, for interoperability between Transport Models and transport-aware Security Models, entries in this cache must include at least the following fields:

- transportDomain
- transportAddress
- tmSecurityName
- tmRequestedSecurityLevel
- tmTransportSecurityLevel
- tmSameSecurity

tmSessionID

4.5.2.1. Transport information

Information about the source of an incoming SNMP message is passed up from the Transport subsystem as far as the Message Processing subsystem. However these parameters are not included in the processIncomingMsg ASI defined in RFC3411, and hence this information is not directly available to the Security Model.

A transport-aware Security Model might wish to take account of the transport protocol and originating address when authenticating the request, and setting up the authorization parameters. It is therefore necessary for the Transport Model to include this information in the tmStateReference cache, so that it is accessible to the Security Model.

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

- transportAddress: the source of the incoming message.

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

4.5.2.2. securityName

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

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

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

- o securityName: a human-readable name (in snmpAdminString format) representing the SNMP principal in a model-independent manner.

The transport principal may or may not be the same as the tmSecurityName. Similarly, the tmSecurityName may or may not be the

same as the securityName as seen by the Application and Access Control subsystems. In particular, a non-transport-aware Security Model will ignore tmSecurityName completely when determining the SNMP securityName.

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

4.5.2.3. securityLevel

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

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

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

4.5.2.4. Session Information

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

specific decision whether to REQUIRE this.

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

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

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

4.5.3. DTLS Transport Model Cached Information

For the DTLS Transport Model, the session state is maintained using tmStateReference. Upon opening each DTLS session, the DTLS Transport Model stores model- and mechanism-specific information about the session in a cache, referenced by tmStateReference. An implementation might store the contents of the cache in a Local Configuration Datastore (LCD).

At a minimum, the following parameters are stored in the cache:

tmTransportDomain = Specified by the application tmSameSecurity = boolean value set by the security model or false by default

tmTransportAddress = Specified by the application

tmRequestedSecurityLevel = ["noAuthNoPriv" | "authNoPriv" | "authPriv"] the security level requested by the application

tmSecurityLevel = ["noAuthNoPriv" | "authNoPriv" | "authPriv"] the security level of the established DTLS session

tmSecurityName = the security name associated with a principal

The tmStateReference cache is used to pass a reference to these values between the DTLS Transport Model and the security model.

The DTLS Transport Model has the responsibility for releasing the complete tmStateReference and deleting the associated information when the session is destroyed.

4.5.3.1. Transport Information

The tmTransportDomain and tmTransportAddress identify the type and address of the remote DTLS transport endpoint. The domain for address types for DTLS sessions SHOULD be "snmpDTLSDomain" and the address SHOULD be one that conforms to the details specified in the "SnmpDTLSAddress" textual convention.

4.5.3.2. tmRequestedSecurityLevel

The tmRequestedSecurityLevel is the security level requested by the application. This parameter is set in the cache by the security

model and used by DTLS Transport Model initiating a session to select the appropriate cipher_suites and other configuration needed settings for establishing the session. The DTLS Transport Model MUST ensure that the actual security provided by the session (tmSecurityLevel) is at least as high as the requested security level (tmRequestedSecurityLevel).

4.5.3.3. tmSecurityLevel

The tmSecurityLevel is the actual security level of the established session. See Section 3.1.2 for more detail about security levels. How the chosen cipher_suites and other DTLS session parameters are translated into a security level at the DTLS Transport Model is implementation dependent and/or policy specific. Implementations MUST NOT use NULL algorithms for fulfilling authentication or encryption needs indicated by the tmSecurityLevel.

4.5.3.4. tmSecurityName

The tmSecurityName is the name of the principal on whose behalf the message is being sent. This field is set via the mapping defined in

the dtlstmCertificateToSNTable when mapping incoming client connection certificates to a tmSecurityName. For outgoing connections, the application will specify the value that should be placed in this field (possibly by extracting it from the SNMP-TARGET-MIB's snmpTargetParamsSecurityName value).

4.5.4. Transport Model LCD

Implementations may store DTLS-specific and model-specific information in a LCD. The DTLS session ID is one such parameter that could be stored in the LCD. When messages are to be routed for encapsulation or for integrity verification and decryption the message and the DTLS session ID must be passed to the DTLS transport layer for processing. Therefore, the DTLS Transport Model MUST maintain a one-to-one mapping between the DTLS session ID and the tmStateReference cache entry for that session. Implementations will need to store the DTLS session ID in the tmStateReference cache to simplify the procedure.

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 DTLSTM 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 with zero value or random value data 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

The following section describes the procedures followed by the DTLS Transport Model when it receives a DTLS protected packet. The steps are broken into two different sections. The first section describes the needed steps for de-multiplexing multiple DTLS sessions and the second section describes the steps which are specific to transport processing once the DTLS processing has been completed.

5.1.1. DTLS Processing for Incoming Messages

DTLS is significantly different in terms of session handling than SSH, TLS or other TCP-based session streams. The DTLS protocol, which is UDP-based, does not have a session identifier that allows implementations to determine through which session a packet is arriving like TCP-based streams have. Thus, a process for de-multiplexing sessions must be incorporated into the procedures for an incoming message. The steps in this section describe how this can be accomplished, although any implementation dependent method for doing so should be suitable as long as the results are consistently deterministic. The important results from the steps in this section are the transportDomain, the transportAddress, the wholeMessage, the wholeMessageLength, and a unique implementation-dependent session identifier.

This procedure assumes that upon session establishment, an entry in a local transport mapping table is created in the Transport Model's LCD. This transport mapping table entry should be able to map a unique combination of the remote address, remote port number, local address and local port number to a implementation-dependent dtlsSessionID.

1) The DTLS 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 DTLS Transport Model queries the LCD using the transport parameters to determine if a session already exists and its dtlsSessionID. As noted previously, the source and destination addresses and ports of the message should uniquely assign the message to a specific session identifier. However, another implementation-dependent method may be used if so desired.

3) If a matching entry in the LCD does not exist then the message is discarded. Increment the dtlstmSessionNoAvailableSessions 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 dtlsSessionID from the LCD.

5) The dtlsWholeMsg, and the dtlsSessionID are passed to DTLS for integrity checking and decryption using the dtlsRead() ASI.

6) If the message fails integrity checks or other DTLS security processing then the dtlstmDTLSProtectionErrors counter is incremented, the message is discarded and processing of the message is stopped.

7) The output of the dtlsRead results in an incomingMessage and an incomingMessageLength. These results and the dtlsSessionID 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 DTLS Transport should process messages that have already been properly extracted from the DTLS stream, as described in Section 5.1.1.

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

   tmTransportDomain = snmpDTLSDomain

   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.

   tmSessionID = A unique session identifier for this DTLS 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.

2) The wholeMessage and the wholeMessageLength are assigned values from the incomingMessage and incomingMessageLength values from the DTLS processing.

3) The DTLS Transport Model passes the transportDomain, transportAddress, wholeMessage, and wholeMessageLength to the Dispatcher using the receiveMessage ASI:

```
statusInformation =
receiveMessage(
IN   transportDomain       -- snmpSSHDomain
IN   transportAddress      -- address for the received message
IN   wholeMessage          -- the whole SNMP message from SSH
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 DTLS 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 DTLS 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 dtlstmSessionNoAvailableSessions 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 3b. 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.

   3a) If there is a corresponding LCD entry, then this session will be used to send the message.

   3b) 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 dtlstmSessionOpenErrors, 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 DTLS for encapsulation and transmission.

5.3. Establishing a Session

The DTLS Transport Model provides the following primitive to establish a new DTLS 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  dtlsSessionID            -- Session identifier for DTLS
)
```

The following sections describe the procedures followed by a DTLS Transport Model when establishing a session as a Command Generator, a Notification Originator or as part of a Proxy Forwarder.

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

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

DTLS 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 UDP/port

combination. Therefore, implementations MAY support responding with multiple identities using separate UDP 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 dtlstmParamsTable 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 DTLS session. However this is done, the security capabilities provided by DTLS 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 DTLS 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 DTLS handshake protocol to establish session keys for message integrity and encryption.

   If the attempt to establish a session is unsuccessful, then dtlstmSessionOpenErrors 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 DTLS server side of the connection identifies the authenticated identity from the DTLS client's principal certificate using the dtlstmCertificateToSNTable mapping table and records this in the tmStateReference cache as

tmSecurityName. If this verification fails in any way (for example because of failures in cryptographic verification or the lack of an appropriate row in the dtlstmCertificateToSNTable) then the session establishment MUST fail, the dtlstmSessionInvalidClientCertificates object is incremented and processing is stopped.

b) The DTLS client side of the connection SHOULD verify that authenticated identity of the DTLS server's certificate is the expected identity and MUST do so if the client application is a Notification Generator. If strong authentication is desired then the DTLS server certificate MUST always be verified and checked against the expected identity. Methods for doing this are described in [I-D.hodges-server-ident-check]. DTLS 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 dtlstmSessionInvalidServerCertificates object is incremented and processing is stopped.

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

5.4. Closing a Session

The DTLS 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 DTLS 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 DTLS 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 A.

6.1. Structure of the MIB Module

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

6.2. Textual Conventions

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

This module defines two new Textual Conventions: a new TransportDomain and TransportAddress format for describing DTLS connection addressing requirements.

6.3. Statistical Counters

The DTLSTM-MIB defines some statical counters that can provide network managers with feedback about DTLS session usage and potential errors that a MIB-instrumented device may be experiencing.

6.4. Configuration Tables

The DTLSTM-MIB defines configuration tables that a manager can use for help in configuring a MIB-instrumented device for sending and receiving SNMP messages over DTLS. 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 DTLS 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 DTLS Transport Model. In particular, it is assumed that an entity implementing the DTLSTM-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].

This MIB module is for managing DTLS Transport Model information.

6.5.1. MIB Modules Required for IMPORTS

The following 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

```
DTLSTM-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
    snmpTargetParamsEntry
      FROM SNMP-TARGET-MIB
    ;

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

                  Chairs:
                  Co-editors:
                                "

    DESCRIPTION  "The DTLS 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
```

```
-- ************************************************

dtlstmNotifications OBJECT IDENTIFIER ::= { dtlstmMIB 0 }
dtlstmObjects       OBJECT IDENTIFIER ::= { dtlstmMIB 1 }
dtlstmConformance   OBJECT IDENTIFIER ::= { dtlstmMIB 2 }

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

snmpDTLSDomain OBJECT-IDENTITY
    STATUS      current
    DESCRIPTION
        "The SNMP over DTLS transport domain. The corresponding
        transport address is of type SnmpDTLSAddress.

        When an SNMP entity uses the snmpDTLSDomain 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
        snmpDTLSDomain is 'dtls'.  This prefix may be used by
        security models or other components to identify what secure
        transport infrastructure authenticated a securityName."

    ::= { snmpDomains yy }


-- RFC Ed.: Please replace the I-D reference with a proper one once it
-- has been published.  Note: xml2rfc doesn't handle refs within artwork

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

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

SnmpDTLSAddress ::= 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 snmpDTLSAddress 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))

-- The dtlstmSession Group

dtlstmSession          OBJECT IDENTIFIER ::= { dtlstmObjects 1 }

dtlstmSessionOpens  OBJECT-TYPE
```

```
    SYNTAX       Counter32
    MAX-ACCESS   read-only
```



```
    STATUS       current
    DESCRIPTION
       "The number of times an openSession() request has been
       executed as an SSH client, whether it succeeded or failed."
    ::= { dtlstmSession 1 }

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

dtlstmSessionOpenErrors  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 SSH client, for any reason."
    ::= { dtlstmSession 3 }


dtlstmSessionNoAvailableSessions  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."
    ::= { dtlstmSession 4 }

dtlstmSessionInvalidClientCertificates OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
```

```
    DESCRIPTION
        "The number of times an incoming session was not established
        on an SSH server because the presented client certificate was
        invalid.  Reasons for invalidation includes, but is not
        limited to, crypographic validation failures and lack of a
        suitable mapping row in the dtlstmCertificateToSNTable."
    ::= { dtlstmSession 5 }

dtlstmSessionInvalidServerCertificates OBJECT-TYPE
```



```
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times an outgoing session was not established
        on an SSH client because the presented server certificate was
        invalid.  Reasons for invalidation includes, but is not
        limited to, crypographic validation failures and an unexpected
        presented certificate identity."
    ::= { dtlstmSession 6 }

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

-- Configuration Objects

dtlstmConfig          OBJECT IDENTIFIER ::= { dtlstmObjects 2 }

-- Certificate mapping

dtlstmCertificateMapping    OBJECT IDENTIFIER ::= { dtlstmConfig 1 }

dtlstmCertificateToSNCount OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
```

```
    STATUS      current
    DESCRIPTION
        "A count of the number of entries in the
        dtlstmCertificateToSNTable"
    ::= { dtlstmCertificateMapping 1 }

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



```
dtlstmCertificateToSNTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF DtlstmCertificateToSNEntry
    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 DTLS/SNMP connection the client's presented
        certificate should be examined and validated based on
        an established trusted CA certificate or self-signed public
        certificate.  This table does not provide
        a mechanism for uploading the certificates as that is expected
        to occur through an out-of-band transfer.

        Once the authenticity of the certificate has been verified,
        this table can be consulted to determine the appropriate
        securityName to identify the remote connection.  This is done
        by comparing the issuer's distinguished name against the
        dtlstmCertDN value.  If a matching entry is found then the
        securityName is selected based on the dtlstmCertMapType,
        dtlstmCertSubject and dtlstmCertSecurityName fields and the
        resulting securityName is used to identify the other side of
        the DTLS connection.
```

```
        Users are encouraged to make use of certificates with
        CommonName fields that can be used as securityNames so that a
        single root CA certificate can allow all child certificate's
        CommonName to map directly to a securityName via a 1:1
        transformation.  However, this table is flexible enough to
        allow for situations where existing an existing deployed
        certificate infrastructures dose not provide adequate
        CommonName values for use as SNMPv3 securityNames."
    ::= { dtlstmCertificateMapping 3 }

dtlstmCertificateToSNEntry OBJECT-TYPE
    SYNTAX      DtlstmCertificateToSNEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A row in the dtlstmCertificateToSNTable that specifies a
        mapping for an incoming DTLS certificate to a securityName to
        use for the connection."
    INDEX   { dtlstmCertID }
    ::= { dtlstmCertificateToSNTable 1 }

DtlstmCertificateToSNEntry ::= SEQUENCE {
```



```
    dtlstmCertID           Unsigned32,
    dtlstmCertIssuerDN     OCTET STRING,
    dtlstmCertMapType      INTEGER,
    dtlstmCertSecurityName SnmpAdminString,
    dtlstmCertStorageType  StorageType,
    dtlstmCertRowStatus    RowStatus
}

dtlstmCertID OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A unique arbitrary number index for a given certificate
        entry."
    ::= { dtlstmCertificateToSNEntry 1 }

dtlstmCertIssuerDN OBJECT-TYPE
```

```
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The issuer's 'distinguished name' field matching the
        certificate to be used for this mapping entry.
        "
--        XXX: allow specification via serial no too?
    ::= { dtlstmCertificateToSNEntry 2 }

dtlstmCertMapType OBJECT-TYPE
    SYNTAX      INTEGER { specified(1), byCN(2) }
    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 locally to
                      identify the remote entity is directly specified
                      in the dtlstmCertSecurityName column from this
                      table.

        byCN(2):      The securityName that should be used locally to
                      identify the remote entity should be taken from
                      the CommonName portion of the Subject field from
                      the X.509 certificate."
    DEFVAL { specified }
    ::= { dtlstmCertificateToSNEntry 3 }
```



```
dtlstmCertSecurityName OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE(0..32))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The securityName that the session should use if the
        dtlstmCertMapType is set to specified(1), otherwise the value
        in this column should be ignored.  If dtlstmCertMapType 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."
    DEFVAL { "" }
    ::= { dtlstmCertificateToSNEntry 4 }

dtlstmCertStorageType 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 }
    ::= { dtlstmCertificateToSNEntry 5 }


dtlstmCertRowStatus 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.

        The value of this object has no effect on whether
        other objects in this conceptual row can be modified."
    ::= { dtlstmCertificateToSNEntry 6 }

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

dtlstmParamsCount OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "A count of the number of entries in the
        dtlstmParamsTable"
```



```
    ::= { dtlstmCertificateMapping 4 }

dtlstmParamsTableLastChanged OBJECT-TYPE
    SYNTAX      TimeStamp
```

```
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of sysUpTime.0 when the dtlstmParamsTable
        was last modified through any means, or 0 if it has not been
        modified since the command responder was started."
    ::= { dtlstmCertificateMapping 5 }

dtlstmParamsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF DtlstmParamsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table augments the SNMP-TARGET-MIB's
        snmpTargetParamsTable with additional a DTLS client-side
        certificate certificate identifier to use when establishing
        new DTLS connections."
    ::= { dtlstmCertificateMapping 6 }

dtlstmParamsEntry OBJECT-TYPE
    SYNTAX      DtlstmParamsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A conceptual row containing the certificate subject name for
        a given snmpTargetParamsEntry.  The values in this row should
        be ignored if not the connection that needs to be established,
        as indicated by the SNMP-TARGET-MIB infrastructure, is not a
        DTLS based connection."
    AUGMENTS    { snmpTargetParamsEntry }
    ::= { dtlstmParamsTable 1 }

DtlstmParamsEntry ::= SEQUENCE {
    dtlstmParamsSubject         OCTET STRING,
    dtlstmParamsStorageType     StorageType,
    dtlstmParamsRowStatus       RowStatus
}

dtlstmParamsSubject OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE(1..4096))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The subject name of the locally-held X.509 certificate that
```

```
        should be used when initiating a DTLS connection as a DTLS
        client."
    ::= { dtlstmParamsEntry 1 }

dtlstmParamsStorageType 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 }
    ::= { dtlstmParamsEntry 2 }


dtlstmParamsRowStatus 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.

        The value of this object has no effect on whether
        other objects in this conceptual row can be modified."
    ::= { dtlstmParamsEntry 3 }

-- ************************************************
-- dtlstmMIB - Conformance Information
-- ************************************************

dtlstmCompliances OBJECT IDENTIFIER ::= { dtlstmConformance 1 }

dtlstmGroups OBJECT IDENTIFIER ::= { dtlstmConformance 2 }



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

dtlstmCompliance MODULE-COMPLIANCE
    STATUS      current
    DESCRIPTION
        "The compliance statement for SNMP engines that support the
        SNMP-DTLS-TM-MIB"
```

```
    MODULE
```



```
        MANDATORY-GROUPS { dtlstmStatsGroup,
                           dtlstmIncomingGroup, dtlstmOutgoingGroup }
    ::= { dtlstmCompliances 1 }

-- ************************************************
-- Units of conformance
-- ************************************************
dtlstmStatsGroup OBJECT-GROUP
    OBJECTS {
        dtlstmSessionOpens,
        dtlstmSessionCloses,
        dtlstmSessionOpenErrors,
        dtlstmSessionNoAvailableSessions,
        dtlstmSessionInvalidClientCertificates,
        dtlstmSessionInvalidServerCertificates,
        dtlstmDTLSProtectionErrors
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects for maintaining
        statistical information of an SNMP engine which
        implements the SNMP DTLS Transport Model."
    ::= { dtlstmGroups 1 }

dtlstmIncomingGroup OBJECT-GROUP
    OBJECTS {
        dtlstmCertificateToSNCount,
        dtlstmCertificateToSNTableLastChanged,
        dtlstmCertIssuerDN,
        dtlstmCertMapType,
        dtlstmCertSecurityName,
        dtlstmCertStorageType,
        dtlstmCertRowStatus
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects for maintaining
        incoming connection certificate mappings to
        securityNames of an SNMP engine which implements the
        SNMP DTLS Transport Model."
```

```
    ::= { dtlstmGroups 2 }

dtlstmOutgoingGroup OBJECT-GROUP
    OBJECTS {
        dtlstmParamsCount,
        dtlstmParamsTableLastChanged,
        dtlstmParamsSubject,
        dtlstmParamsStorageType,
```



```
        dtlstmParamsRowStatus
    }
    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."
    ::= { dtlstmGroups 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 DTLS client and the DTLS server. State information for the sessions are maintained in each DTLSTM 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, DTLS clients will always send REQUEST messages and DTLS 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.

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, a time-to-live (TTL) value SHOULD be established for sessions. An upper limit of 24 hours is suggested for this TTL value. The TTL value could be stored in the LCD and checked before passing a message to the DTLS session.

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 DTLS 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 DTLS transport 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 DTLS security services. The security threats and how the DTLS 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 Appendices D, E, and F of TLS 1.1 [RFC4346]. 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.

DTLS provides for both authentication of the identity of the DTLS server and authentication of the identity of the DTLS 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, as discussed in Section 4.5.3.4, Section 7 and other sections.

Furthermore, the DTLS handshake only provides assurance that the certificate of the authenticated identity has been signed by an configured accepted Certificate Authority. DTLS 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 DTLS 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 DTLS server and the DTLS client for all operations ensures the authenticity of the SNMP engine that provides MIB data.

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 DTLS 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 DTLS Transport Model.

9.3. MIB Module Security

The MIB objects in this document should be protected with an adequate level of at least integrity protection, especially those objects which are writable. Since knowledge of authorization and certificate usage mechanisms may be considered sensitive, protection from disclosure of the SNMP traffic via encryption is also recommended.

SNMP versions prior to SNMPv3 did not include adequate security.

Even if the network itself is secure (for example by using IPSec or DTLS) 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 section 8 of [RFC3410]), including full support for the USM (see [RFC3414]) and the DTLS Transport Model cryptographic mechanisms (for authentication and privacy).

10. IANA Considerations

IANA is requested to assign:

1. 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 over a DTLS Transport Model as defined in this document,

2. 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 SNMPTRAP over a DTLS Transport Model as defined in this document,

3. an SMI number under snmpDomains for the snmpDTLSDomain object identifier,

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

5. "dtls" as the corresponding prefix for the snmpDTLSDomain in the SNMP Transport Model registry;

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 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.

[RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, March 1999.

[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.

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

[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.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

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

[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)".

[X509] Rivest, R., Shamir, A., and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems".

[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".

12.2. Informative References

[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.hodges-server-ident-check] Hodges, J. and B. Morgan, "Generic Server Identity Check".

Appendix A. 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 DTLSTM 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):


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

Note to RFC editor: replace XXX 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 dtlstmParamsTable. The dtlstmParamsTable 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 DTLS session. The mapping from a recevied DTLS client certificate to a securityName is done with the dtlstmCertificateToSNTable. The certificates must be loaded into the device so that a dtlstmCertificateToSNEntry may refer to it. As an example, consider the following entry which will provide a mapping from a X.509 Issuer's Distinguished Name directly to the "blueberry" securityName:

```
  dtlstmCertID           = 1         (arbitrarily chosen)
  dtlstmCertIssuerDN     = "C=US, ST=California, ..., CN=hardaker"
  dtlstmCertMapType      = specified(1)
  dtlstmCertSecurityName = "blueberry"
  dtlstmCertStorageType  = 3 (nonVolatile)
  dtlstmCertRowStatus    = 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 CommonName mappings where possible since it will provide a more scalable approach to certificate management. If the following entry was created:


```
  dtlstmCertID          = 1         (arbitrarily chosen)
  dtlstmCertIssuerDN    = "C=US, ST=California, L=Davis, O=SuprIDs, ..."
  dtlstmCertMapType     = byCN(2)
  dtlstmCertStorageType = 3 (nonVolatile)
  dtlstmCertRowStatus   = 4 (createAndGo)
```

The above entry indicates the CommonName field for that particular Issuer 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 CA is carefully controlled.

For the example, if the incoming DTLS client provided certificate contained a Subject with a CommonName of "blueberry" and the certificate was signed by the CA matching the dtlstmCertIssuerDN value above and the CA's certificate was properly installed on the device then the CommonName of "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

Full Copyright Statement

Intellectual Property

pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.