Secure Shell Security Model for SNMP
draft-harrington-isms-secshell-01.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

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

This Internet-Draft will expire on March 20, 2006.

Copyright Notice

Copyright (C) The Internet Society (2005).

Abstract

This memo describes a Security Model for the Simple Network Management Protocol, using the Secure Shell protocol within a Transport Mapping.
7.  Security Considerations .......................... 40
8.  IANA Considerations ................................. 40
9.  Acknowledgments .................................. 40
10. References .......................................... 40
10.1 Normative References ............................... 40
10.2 Informative References ............................. 42
A.  Change Log from the first revision of -00- ........... 43
    Intellectual Property and Copyright Statements ....... 44
1. Introduction

This memo describes a Security Model for the Simple Network Management Protocol, using the Secure Shell protocol within a Transport Mapping.

It is important to understand the SNMP architecture and the terminology of the architecture to understand where the Security Model described in this memo fits into the architecture and interacts with other subsystems within the architecture. The reader is expected to have read and understood the description of the SNMP architecture, as defined in [RFC3411], and the "Transport Mapping Security Model (TMSM) for the Simple Network Management Protocol" architecture extension defined in [TMSM], which enables the use of external "lower layer" protocols to provide message security, tied into the SNMP architecture through the transport mapping subsystem. One such external protocol is the Secure Shell protocol [SSHArch].

This memo describes the Secure Shell Security Model for SNMP, a specific SNMP security model to be used within the SNMP Architecture, to provide authentication, encryption, and integrity checking of SNMP messages.

This memo defines a portion of the Management Information Base (MIB) for use with network management protocols in TCP/IP based internets. In particular it defines objects for monitoring and managing the Secure Shell Security Model for SNMP.

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

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

1.1 Motivation

Version 3 of the Simple Network Management Protocol (SNMPv3) added security to the previous versions of the protocol. The User Security Model (USM) [RFC3414] was designed to be independent of other existing security infrastructures, to ensure it could function when
third party authentication services were not available, such as in a broken network. As a result, USM typically utilizes a separate user and key management infrastructure. Operators have reported that deploying another user and key management infrastructure in order to use SNMPv3 is a reason for not deploying SNMPv3 at this point in time.

This memo describes a security model that will make use of the existing and commonly deployed Secure Shell security infrastructure. It is designed to meet the security and operational needs of network administrators, maximize useability in operational environments to achieve high deployment success and at the same time minimize implementation and deployment costs to minimize the time until deployment is possible.

The work will address the requirement for the SSH client to authenticate the SSH server, for the SSH server to authenticate the SSH client (the user), and how SNMP can make use of the authenticated identities in authentication and auditing.

The work will include the ability to use any of the user authentication methods described in "SSH Authentication Protocol" [SSHAuth] - public key, password, and host-based. Local accounts may be supported through the use of the public key, host-based or password based mechanisms. The password based mechanism allows for integration with deployed password infrastructure such as AAA servers using the RADIUS protocol [RFC2865]. In the future it should also be able to take advantage of other defined authentication mechanism such as those defined in [gsskeyex] which will allow for user authentication mechanisms which support different security infrastructures and provide security properties.

It is desirable to use mechanisms that could unify the approach for administrative security for SNMPv3 and CLI and other management interfaces. The use of security services provided by Secure Shell is the approach commonly used for the CLI, and is the approach being adopted for use with NETCONF [Netconf]. Similar to NETCONF over SSH [NetconfSSH], this memo describes a method for invoking and running the SNMP protocol within a Secure Shell (SSH) session as an SSH subsystem.

This memo defines how SNMP can be used within a Secure Shell (SSH) session, using the SSH connection protocol [SSHConnect] over the SSH transport protocol [SSHTrans], using SSH user-auth [SSHAuth] for authentication.

There are a number of challenges to be addressed to map Secure Shell authentication method parameters into the SNMP architecture so that
SNMP continues to work without any surprises. These are discussed in detail below. Some points requiring further WG research and discussion are identified by [todo] markers in the text.

1.2 The Internet-Standard Management Framework

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

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB module that is compliant to the SMIV2, which is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

1.3 The Secure Shell Protocol

SSH is a protocol for secure remote login and other secure network services over an insecure network. It consists of three major components:

- The Transport Layer Protocol [[SSHTrans] provides server authentication, confidentiality, and integrity. It may optionally also provide compression. The transport layer will typically be run over a TCP/IP connection, but might also be used on top of any other reliable data stream.
- The User Authentication Protocol [SSHAuth] authenticates the client-side user to the server. It runs over the transport layer protocol.
- The Connection Protocol [SSHConnect] multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol.

The client sends a service request once a secure transport layer connection has been established. A second service request is sent after user authentication is complete. This allows new protocols to be defined and coexist with the protocols listed above.

The connection protocol provides channels that can be used for a wide range of purposes. Standard methods are provided for setting up secure interactive shell sessions and for forwarding ("tunneling") arbitrary TCP/IP ports and X11 connections.
1.4 Constraints

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

1. When the requirements of effective management in times of network stress are inconsistent with those of security, the design of this model gives preference to the former.

2. In times of network stress, neither the security protocol nor its underlying security mechanisms should depend upon the ready availability of other network services (e.g., Network Time Protocol (NTP) or AAA protocols).

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

4. It may not be possible for the security model to determine when the network is under stress.

5. A security mechanism should entail no changes to the basic SNMP network management philosophy.

1.5 Conventions

The terms "manager" and "agent" are not used in this document, because in the RFC 3411 architecture, all entities have the capability of acting as either manager or agent or both depending on the SNMP applications included in the engine. Where distinction is required, the application names of Command Generator, Command Responder, Notification Generator, Notification Responder, and Proxy Forwarder are used. See "SNMP Applications" [RFC3413] for further information.

Throughout this document, the terms "client" and "server" are used to refer to the two ends of the SSH transport connection. The client actively opens the SSH connection, and the server passively listens for the incoming SSH connection. Either entity may act as client or as server, as discussed further below.

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

2. How SSHSM Fits into the TMSM Architecture

SSH is a security layer which is plugged into the TMSM architecture between the underlying transport layer and the message dispatcher.

The SSHSM model will establish an encrypted tunnel between the transport mappings of two SNMP engines. The sending transport mapping security model instance encrypts outgoing messages, and the
receiving transport mapping security model instance decrypts the messages.

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

Within an engine, outgoing SNMP messages are passed unencrypted from the message dispatcher to the transport mapping, and incoming messages are passed unencrypted from the transport mapping to the message dispatcher. SSHSM security processing will be called from within the Transport Mapping functionality of an SNMP engine dispatcher to perform the translation of transport security parameters to/from security-model-independent parameters. Some SSHSM security processing will also be performed within a message processing portion of the model, for compatibility with the ASIs between the RFC 3411 Security Subsystem and the Message Processing Subsystem.

2.1 Security Capabilities of this Model

2.1.1 Threats

The security protocols used in this memo are considered acceptably secure at the time of writing. However, the procedures allow for new authentication and privacy methods to be specified at a future time if the need arises.

The Secure Shell Security Model provides protection against the threats identified by the RFC 3411 architecture [RFC3411]:

1. Message stream modification - Provide for verification that each received SNMP message has not been modified during its transmission through the network.
2. Information modification - Provide for verification that the contents of each received SNMP message has not been modified during its transmission through the network. Data has not been altered or destroyed in an unauthorized manner, nor have data sequences been altered to an extent greater than can occur non-maliciously.
3. Masquerade - Provide for both verification of the identity of the user on whose behalf a received SNMP message claims to have been generated, and the verification of the identity of the MIB owner. For the protocols specified in this memo, it is not possible to assure the specific user that originated a received SNMP message; rather, it is the user on whose behalf the message was originated.
that is authenticated. SSH provides verification of the identity of the MIB owner through the SSH Transport Protocol server authentication [SSHTrans].

4. Verification of user identity is important for use with the SNMP access control subsystem, to ensure that only authorized users have access to potentially sensitive data. The SSH user identity will be used to map to an SNMP model-independent securityname for use with SNMP access control.

5. Authenticating the server ensures the authenticity of the SSH server that is associated with the SNMP engine that provides MIB data. Operators or management applications could act upon the data they receive (e.g. raise an alarm for an operator, modify the configuration of the device that sent the notification, modify the configuration of other devices in the network as the result of the notification, and so on), so it is important to know that the data is authentic. SSH allows for authentication of the SSH server using the SSH public key credentials described in [SSHTrans] and mechanisms such as those described in [gsskeyex].

6. Disclosure - Provide, when necessary, that the contents of each received SNMP message are protected from disclosure to unauthorized persons.

7. Replay - Provide for detection of received SNMP messages, which request or contain management information, whose time of generation was not recent. A message whose generation time is outside of a time window is not accepted. Note that message reordering is not dealt with and can occur in normal conditions.

2.1.1.1 Data Origin Authentication Issues

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)

SSH provides support for encryption and data integrity. While it is technically possible to support noAuthNoPriv and authNoPriv in SSH it is NOT RECOMMENDED by [SSHTrans]. This means that an SSH connection should support authPriv, which is the highest level of security defined in RFC 3411. It is possible for SSH to skip entity authentication of the client through the "none" authentication method to support anonymous clients, however in this case an implementation MUST still support data integrity within the SSH transport protocol. The security protocols used in [SSHTrans] are considered acceptably secure at the time of writing. However, the procedures allow for new authentication and privacy methods to be specified at a future time if the need arises.
authNoPriv may be important to accommodate governmental regulation (e.g. export laws) regarding encryption technologies.

[todo] is it important to support anonymous user access to SNMP?

Should the transport layer provide what data integrity and encryption algorithms were negotiated to the SSHSM layer? In SNMP, we deliberately avoided this, and settled for an assertion that auth and priv were applied according to the rules of the security model.

SSH should also provide the identity of the authenticated parties. From this information it should be possible for the SNMP subsystem to determine if the session is allowed access to the subsystem.

2.1.1.1.1 noAuthPriv

SSH provides the "none" userauth method, which is normally rejected by servers and used only to find out what userauth methods are supported. However, it is legal for a server to accept this method, which has the effect of not authenticating the ssh client to the ssh server. Doing this does not compromise authentication of the ssh server to the ssh client, nor does it compromise data confidentiality or data integrity.

The RFC 3411 architecture does not permit noAuthPriv. SSHSM should refuse a noAuthPriv session [todo] If we do not allow some of these options, how do we determine the option was used, so we can reject it? How does an SNMP engine reject a session?

2.1.1.1.2 skipping public key verification

[todo] Most key exchange algorithms are able to authenticate the SSH server’s identity to the client. However, for the common case of DH signed by public keys, this requires the client to know the host’s public key a priori and to verify that the correct key is being used. If this step is skipped, you no longer have authentication of the ssh server to the ssh client. You do still get data confidentiality and data integrity protection to whatever server you’re talking to, but these are of dubious value when an attacker can insert himself between the client and the real ssh server. Note that some userauth methods may defend against this situation, but many of the common ones (including password and keyboard-interactive) do not, and in fact depend on the fact that the server’s identity has been verified (otherwise you may be giving your password to an attacker).

2.1.1.1.3 the ‘none’ MAC algorithm

SSH provides the "none" MAC algorithm, which would allow you to turn
of data integrity while maintaining confidentiality. However, if you do this, then an attacker may be able to modify the data in flight, which means you effectively have no authentication.

SSH must not be configured using the "none" MAC algorithm for use with the SSHSM security model.

2.1.2 Sessions

Sessions are not part of RFC 3411 architecture, but are considered desirable because the cost of authentication can be amortized over potentially many transactions. The Secure Shell security model will utilize sessions, with a single user and security level associated with each session. If an exchange with another engine would require a different security level or would be on behalf of a different user, then another session would be needed. An immediate consequence of this is that implementations should be able to maintain some reasonable number of concurrent sessions. This document will discuss the impact of sessions on SNMP usage. [todo]

2.1.2.1 Message security versus session security

As part of session creation the client and server entities are typically authenticated and authorized access to the session. In addition as part of session establishment cryptographic key material is exchanged and is then used to control access to the session on a message by message basis. Messages that fail the basic data origin authentication/ data integrity checks will be rejected. Entities receiving the messages that do not have the correct encryption keys established during session creation will not be able to read the messages. In order for an entity to process messages it must maintain certain state associated with the session. This includes, but is not limited to cryptographic encryption and data integrity keys, entity identities and authorization information associated with the authenticated identities. After a message is received and passes integrity and authentication checks then the state stored in the session is used to provide further authorization for the message.

2.1.3 Authentication Protocol

SSHSM should support any user authentication mechanism supported by SSH.

The SSH Authentication Protocol document describes three authentication methods - publickey, password, and host-based. All three authentication methods are supported by the Secure Shell Security Model for SNMP.
The password authentication mechanism allows for integration with deployed password based infrastructure. It is possible to hand a password to a service such as RADIUS [RFC2865] or Diameter [RFC 3588] for validation. The validation could be done using the user-name and user-password attributes. It is also possible to use a different password validation protocol such as CHAP [RFC1994] or digest authentication [RFC 2617, draft-ietf-radext-digest-auth-04] to integrate with RADIUS. Any of these mechanism leave the password in the clear on the device that is authenticating the password which introduces threats on the authentication infrastructure which is less than ideal. It is possible that new mechanism will be developed using authentication mechanisms defined in [gsskeyex] which will allow for user authentication mechanisms which support different security infrastructures and provide security properties.

2.1.4 Privacy Protocol

The Secure Shell Security Model uses the SSH transport layer protocol, which provides strong encryption, server authentication, and integrity protection.

2.1.5 Protection against Message Replay, Delay and Redirection

The Secure Shell Security Model uses the SSH transport layer protocol. SSH uses sequence numbers and integrity checks to protect against replay and reordering of messages within a connection.

SSH also provides protection against replay of entire sessions. In a properly-implemented DH exchange, both sides will generate new random numbers for each exchange, which means the exchange hash and thus the encryption and integrity keys will be distinct for every session. This would prevent capturing an SNMP message and redirecting it to another SNMP engine.

Message delay is not as important an issue with SSH as it is with USM. USM checks the timeliness of messages because it does not provide session protection or message sequence ordering. The only delay that would seem to be possible would be to capture the next sequence numbered packet and hold it to play within the same session later.

2.1.6 Security Protocol Requirements

Modifying the Secure Shell protocol, or configuring it in a particular manner, may change its security characteristics in ways that would impact other existing usages. If a change is necessary, the change should be an extension that has no impact on the existing
usages. This document will describe the use of an SSH subsystem for SNMP.

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

There has been discussion of ways SNMP could be extended to better support management/monitoring needs when a network is running just fine. Use of a TCP transport, for example, could enable larger message sizes and more efficient table retrievals. Secure Shell runs over TCP. This document will discuss the expected ramifications of using a TCP transport for SNMP, and the coexistence of UDP and TCP transport for SNMP. Should this be discussed in the TMSM document?

The Secure Shell security model can coexist with the USM security model, the only other currently defined security model. compare to RFC3584 to see if there are any wrinkles to coexistence with SNMPv1/v2c.

2.1.6.1 Mapping SSH to EngineID

In the RFC3411 architecture, there are three use cases for an engineID:

- **snmpEngineID** - RFC3411 includes the SNMP-FRAMEWORK-MIB, which defines a snmpEngineID object. An snmpEngineID is the unique and unambiguous identifier of an SNMP engine. Since there is a one-to-one association between SNMP engines and SNMP entities, it also uniquely and unambiguously identifies the SNMP entity within an administrative domain.

- **contextEngineID** - Management information resides at an SNMP entity where a Command Responder Application has local access to potentially multiple contexts. This application uses a contextEngineID equal to the snmpEngineID of its associated SNMP engine.

- **securityEngineID** - The RFC3411 architecture defines ASIs that include a securityEngineID - the authoritative SNMP entity - which is either the local snmpEngineID or the target snmpEngineID,
depending on the type of operation. Since a security model might utilize shared credentials and integrity-checking parameters, and the datastores of the two endpoints could get out of sync, the "authoritative" engineID indicates which end has the values to be used.

The securityEngineID is used by USM when performing integrity checking and authentication, to look up values in the USM tables, and to synchronize "clocks". The securityEngineID is not needed by SSHSM, since integrity checking and authentication are handled outside the SNMP engine.

[todo] is there still a need for an "authoritative SNMP engine"? Does authoritative have any meaning in a TMSM/SSHSM environment? In SNMPv3, the authoritative engine is usually the engine with the command responder, i.e. the agent; in non-proxy situations, securityEngineID equals contextEngineID. In client-server terms, the authoritative engine is usually the server. So, should the SNMP engine associated with the SSH server be authoritative? Would Infoms change that? Would bidirectional messaging change that? Would call-home change that? Do we need to set the securityEngineID to indicate which side is the SSH server?

2.2 Security Parameter Passing Requirement

[SSHSM follows the TMSM approach, in which the security-model specific parameters can be determined from the transport layer by the transport mapping, before the message processing begins.

[todo] For outgoing messages, it is necessary to have an MPSP portion of the security model because it is the MPSP that actually creates the WholeMsg from its component parts. In the SSHSM model, the MPSP does not apply encryption, integrity-checking, or authentication. For example, an SNMPv3 message is built without any content in the SecurityParameters field, and the WholeMsg is passed unencrypted back to the Message Processing Model for forwarding to the Transport Mapping.

A cache mechanism will be used, into which the TMSP puts information about the security applied to an incoming message, and an MPSP extracts that information from the cache. Given that there may be multiple TM-security caches, a cache ID will need to be passed through an ASI so the MPSP knows which cache of information to consult. [todo]

The cache reference could be thought of as an additional parameter in the ASIs between the transport mapping and the messaging security model. The RFC 3411 ASIs would not need to be changed since the
SNMPv3 WG expected that additional parameters could be passed for value-add features of specific implementations.

This approach does create dependencies between a model-specific TPSP and a corresponding specific MPSP. If a TMSM-model-independent ASI parameter is passed, this approach would be consistent with the securityStateReference cache already being passed around in the ASI.

2.3 Requirements for Notifications

[todo] cleanup this section

RFC 3430 (SNMP over TCP) suggests that TCP connections are initiated by notification originators in case there is no currently established connection that can be used to send the notification. Following this approach with ssh would require to provision authentication credentials on the agent so that agents can successfully authenticate to a notification receiver. There might be other approaches, like the reuse of manager initiated secure transport connections for notifications. There is some text in Appendix A in RFC 3430 which captures some of these discussions when RFC 3430 was written.

2.4 Scenario Diagrams

RFC 3411 section 4.6 provides scenario diagrams to illustrate how an outgoing message is created, and how an incoming message is processed. Both diagrams are incomplete, however. In section 4.6.1, the diagram doesn’t show the ASI for sending an SNMP request to the network or receiving an SNMP response message from the network. In section 4.6.2, the diagram doesn’t illustrate the interfaces required to receive an SNMP message from the network, or to send an SNMP message to the network.

2.4.1 Command Generator or Notification Originator

This diagram from RFC 3411 4.6.1 shows how a Command Generator or Notification Originator application requests that a PDU be sent, and how the response is returned (asynchronously) to that application.
2.4.2 Command Responder

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Dispatcher</th>
<th>Message Processing</th>
<th>Security Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responder</td>
<td>Dispatcher</td>
<td>Model</td>
<td></td>
</tr>
<tr>
<td>registerContextEngineID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;-------------------</td>
<td>-----------------</td>
<td>--------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>Receive SNMP Message</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>from Network</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>&lt;---------------+</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>prepareDataElements</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>-------------------&gt;</td>
<td>processIncomingMsg</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>&lt;---------------+</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>processPdu</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>&lt;---------------+</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>returnResponsePdu</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>-------------------&gt;</td>
<td>prepareResponseMsg</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>generateResponseMsg</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>Send SNMP Message to Network</td>
</tr>
</tbody>
</table>

### 2.5 Abstract Service Interfaces
3. RFC 3411 Abstract Service Interfaces

Abstract service interfaces have been defined by RFC 3411 to describe the conceptual data flows between the various subsystems within an SNMP entity. The Secure Shell Security Model uses some of these conceptual data flows when communicating with other subsystems, such as the Message Processing Subsystem. These RFC 3411-defined data flows are referred to here as public interfaces.

3.1 Public Abstract Service Interfaces

3.1.1 Public ASIs for Outgoing Messages

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

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

The abstract service primitive from a Message Processing Model to a Security Model to generate the components of a Request message is:
statusInformation = -- success or errorIndication
generateRequestMsg(
  IN messageProcessingModel  -- typically, SNMP version
  IN globalData              -- message header, admin data
  IN maxMessageSize          -- of the sending SNMP entity
  IN securityModel           -- for the outgoing message
  IN securityEngineID        -- authoritative SNMP entity
  IN securityName            -- on behalf of this principal
  IN securityLevel           -- Level of Security requested
  IN scopedPDU               -- message (plaintext) payload
  OUT securityParameters     -- filled in by Security Module
  OUT wholeMsg               -- complete generated message
  OUT wholeMsgLength         -- length of generated message
)

The abstract service primitive from a Message Processing Model to a Security Model to generate the components of a Response message is:

statusInformation = -- success or errorIndication
generateResponseMsg(
  IN messageProcessingModel  -- typically, SNMP version
  IN globalData              -- message header, admin data
  IN maxMessageSize          -- of the sending SNMP entity
  IN securityModel           -- for the outgoing message
  IN securityEngineID        -- authoritative SNMP entity
  IN securityName            -- on behalf of this principal
  IN securityLevel           -- Level of Security requested
  IN scopedPDU               -- message (plaintext) payload
  IN securityStateReference  -- reference to security state information from original request
  OUT securityParameters     -- filled in by Security Module
  OUT wholeMsg               -- complete generated message
  OUT wholeMsgLength         -- length of generated message
)

The abstract data elements passed as parameters in the abstract service primitives are as follows: [todo] check each parameter and determine if it is necessary for SSHSM and whether the description is accurate:

statusInformation - An indication of whether the encoding and securing of the message was successful. If not it is an indication of the problem.
messageProcessingModel - The SNMP version number for the message to be generated. This data is not used by the User-based Security module.
globalData - The message header (i.e., its administrative information). This data is not used by the User-based Security module.
maxMessageSize - The maximum message size as included in the message. This data is not used by the User-based Security module.
securityParameters - These are the security parameters. They will be filled in by the SSH Security module.
securityModel - The securityModel in use. Should be SSH Security Model.

securityName - identifies a principal to be used for securing an outgoing message. The securityName has a format that is independent of the Security Model. In case of a response this parameter is ignored and the value from the cache is used.
securityLevel - The Level of Security from which the SSH Security module determines if the message needs to be protected from disclosure and if the message needs to be authenticated.
securityEngineID - The snmpEngineID of the authoritative SNMP engine to which a dateRequest message is to be sent. In case of a response it is implied to be the processing SNMP engine’s snmpEngineID and so if it is specified, then it is ignored.

scopedPDU - The message payload. The data is opaque as far as the SSH Security Model is concerned.
securityStateReference - A handle/reference to cachedSecurityData to be used when securing an outgoing Response message. This is the exact same handle/reference as it was generated by the SSH Security module when processing the incoming Request message to which this is the Response message.
wholeMsg - The fully encoded SNMP message ready for sending on the wire.
wholeMsgLength - The length of the encoded SNMP message (wholeMsg).

Upon completion of the process, the SSH Security module returns statusInformation. If the process was successful, the completed message is returned, without the privacy and authentication applied yet. If the process was not successful, then an errorIndication is returned.

3.1.2 Public ASIs for Incoming Messages

The abstract service primitive from a Transport Mapping (in the dispatcher) to a Message Processing Model for a received message is::
The abstract service primitive from a Message Processing Model to the Security Subsystem for a received message is:

\[
\text{statusInformation} = \begin{cases} 
\text{errorIndication or success} & \text{error counter OID/value if error} \\
\text{error counter OID/value if error} & \text{error counter OID/value if error} 
\end{cases}
\]

\[
\text{processIncomingMsg}( \\
\text{IN messageProcessingModel} & \text{typically, SNMP version} \\
\text{IN maxMessageSize} & \text{of the sending SNMP entity} \\
\text{IN securityParameters} & \text{for the received message} \\
\text{IN securityModel} & \text{for the received message} \\
\text{IN securityLevel} & \text{Level of Security} \\
\text{IN wholeMsg} & \text{as received on the wire} \\
\text{IN wholeMsgLength} & \text{length as received on the wire} \\
\text{OUT securityEngineID} & \text{authoritative SNMP entity} \\
\text{OUT securityName} & \text{identification of the principal} \\
\text{OUT scopedPDU,} & \text{message (plaintext) payload} \\
\text{OUT maxSizeResponseScopedPDU} & \text{maximum size sender can handle} \\
\text{OUT securityStateReference} & \text{reference to security state} \\
\end{cases}
\]

\[
\text{-- information, needed for response}
\]

3.2 Private Abstract Service Interfaces

A set of abstract service interfaces have been defined within this
document to describe the conceptual data flows between the Secure Shell Security Model (SSHSM) and the self-contained transport mapping services. These apply only to the Secure Shell Security Model (SSHSM), and are referred to here as private interfaces.

The Secure Shell Security Model provides the following internal primitives to pass data back and forth between the Security Model itself and the SSH authentication service:

\[
\text{statusInformation} = \\
\text{establishSession}( \\
\text{IN transportDomain} \quad \text{-- transport domain to be used} \\
\text{IN transportAddress} \quad \text{-- transport address to be used} \\
\text{IN securityModel} \quad \text{-- Security Model to use} \\
\text{IN securityEngineID} \quad \text{-- SNMP entity} \\
\text{IN securityName} \quad \text{-- on behalf of this principal} \\
\text{IN securityLevel} \quad \text{-- Level of Security requested} \\
\text{OUT sessionID} \\
) \\
\]

4. SNMP Messages Using this Security Model

The syntax of an SNMP message using this Security Model adheres to the message format defined in the version-specific Message Processing Model document (for example \[RFC3412\]). At the time of this writing, there are three defined message formats - SNMPv1, SNMPv2c, and SNMPv3.

4.1 SNMPv1 and SNMPv2c Messages Using this Security Model

Since message security is provided by a "lower layer", the message does not need to carry message security parameters within the PDU.

The securityModel and securityName parameters are determined by the Secure Shell Security Model from the SSH service. SSHSM requires that transport always be authenticated and integrity-checked, and encrypted, so all SSHSM messages are authpriv. Since an incoming SNMPv1 or SNMPv2c message lacks a msgFlags field, the msgFlags is always treated as authPriv. [todo]

The communityString is not used as an authentication mechanism, since user authentication is provided by SSH userauth. The community string is still used to provide context information.
The SNMPv1 and SNMPv2c message formats do not contain a contextEngineID, but do contain an IP Address field that can be used to perform proxy. [todo] determine the proxy forwarding mechanism, if any.

4.2 SNMPv3 Messages Using this Security Model

RFC 3412 defines two primitives, generateRequestMsg() and processIncomingMsg() which require the specification of an authoritative SNMP entity. [todo] We need to discuss what the meaning of authoritative would be in a TMSM environment, whether the specific services provided in USM security from msgSecurityParameters are needed in TMSM/SSHSM, and how the Message Processing model provides this information to the security model via generateRequestMsg() and processIncomingMsg() primitives.

The SNMPv3Message SEQUENCE is defined in [RFC3412]. The following fields are specific to the Secure Shell Security Model:
SNMPv3Message Syntax DEFINITIONS IMPLICIT TAGS ::= BEGIN

SNMPv3Message ::= SEQUENCE {
    -- identify the layout of the SNMPv3Message
    -- this element is in same position as in SNMPv1
    -- and SNMPv2c, allowing recognition
    -- the value 3 is used for snmpv3
    msgVersion INTEGER ( 0 .. 2147483647 ),
    -- administrative parameters
    msgGlobalData HeaderData,
    -- security model-specific parameters
    -- format defined by Security Model
    msgSecurityParameters OCTET STRING,
    msgData  ScopedPduData
}

HeaderData ::= SEQUENCE {
    msgID      INTEGER (0..2147483647),
    msgMaxSize INTEGER (484..2147483647),
    msgFlags   OCTET STRING (SIZE(1)),
    -- .... ....1  authFlag
    -- .... ..1.  privFlag
    -- .... .1..  reportableFlag
    -- Please observe:
    -- .... ..00  is OK, means noAuthNoPriv
    -- .... ..01  is OK, means authNoPriv
    -- .... ..10  reserved, MUST NOT be used.
    -- .... ..11  is OK, means authPriv
    msgSecurityModel INTEGER (1..2147483647)
}

ScopedPduData ::= CHOICE {
    plaintext    ScopedPDU,
    encryptedPDU OCTET STRING  -- encrypted scopedPDU value
}

ScopedPDU ::= SEQUENCE {
    contextEngineID OCTET STRING,
    contextName     OCTET STRING,
    data             ANY -- e.g., PDUs as defined in [RFC3416]
}

END
4.2.1 msgGlobalData

SSHSM requires that transport always be authenticated, integrity-checked, and encrypted, so all SSHSM messages are authpriv. The msgFlags MUST always be set to authPriv.


4.2.1.1 msgSecurityParameters

Since message security is provided by a "lower layer", and the securityName parameter is always determined from the SSH authentication method, the SNMP message does not need to carry message security parameters within the msgSecurityParameters field. To prevent its being used in a manner that could be damaging, such as for carrying a virus or worm, when used with SSHSM, it is an empty field. [todo] should this be an ASN.1 NULL within the OCTET STRING, or just a zero-length OCTET STRING?

The field msgSecurityParameters in SNMPv3 messages has a data type of OCTET STRING. Its value MUST be the BER serialization of the following ASN.1 sequence:

SSHMSecurityParametersSyntax DEFINITIONS IMPLICIT TAGS ::= BEGIN
  SSHmssecurityParameters ::= SEQUENCE { NULL }
END

4.2.1.2 msgFlags

[todo] For an outgoing message, msgFlags is the requested security for the message; if a SSHSM cannot provide the requested securityLevel, the request MUST be discarded and SHOULD notify the message processing model that the request failed. [todo: how does this apply in the SSHSM model, especially if the msgFlags MUST always be set to authpriv?]

[todo] do we need do discuss the rest of this, or is this applicable to all TMSM models?

For an outgoing message, it is acceptable for the SSHSM to provide stronger than requested security. To avoid the need to mess with the
ASN.1 encoding, the SNMPv3 message carries the requested msgFlags, not the actual securityLevel applied to the message. If a message format other than SNMPv3 is used, then the new message may carry the more accurate securityLevel in the SNMP message.

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

Part of the responsibility of the SSHSM is to ensure that the actual security provided by the underlying transport layer security mechanisms is configured to meet or exceed the securityLevel required by the msgFlags in the SNMP message. When the MPSP processes the incoming message, it should compare the msgFlags field to the securityLevel actually provided for the message by the transport layer security. If they differ, the MPSP should determine whether the changed securityLevel is acceptable. If not, it should discard the message.

4.3 Passing Security Parameters

For each message received, the Security Model caches the state information such that a Response message can be generated using the same security information, even if the Configuration Datastore is altered between the time of the incoming request and the outgoing response. For SSHSM, there are three levels of state that need to be maintained: the session, the message, and the model-independent translations.

[todo]ensuring consistent security for responses even if the datastore changes is important for USM because USM handles keychanges; will SSHSM allow keychanges to the SSH local datastore? if not, this cache of message-pair-state may be an unnecessary constraint. is it important to ensure responses use the same security as the request for security reasons?

tmSessionReference is used to pass model- and mechanism-specific parameters to coordinate the session-related activities of the TMSP and MPSP. The SSHSM has the responsibility for explicitly releasing the tmSessionReference when the session is destroyed.

tmStateReference is used to pass model- and mechanism-specific parameters to coordinate the activities of the TMSP and MPSP related to a specific pair of messages. The SSHSM has the responsibility for explicitly releasing the tmStateReference once a response message has
been sent, or the data is no longer needed.

The MPSP translates select parameters from the tmSessionReference cache into model-independent parameters subsequently passed in the securityStateReference cache to a Message Processing Model. The Message Processing Model has the responsibility for explicitly releasing the securityStateReference if such data is no longer needed. The securityStateReference cached data may be implicitly released via the generation of a response, or explicitly released by using the stateRelease primitive, as described in RFC 3411 section 4.5.1.

4.3.1 Transport Session Parameters

[todo] SSHSM will create a session between the TMSM of one SNMP entity and the TMSM of another SNMP entity. The created "tunnel" will provide encryption and data integrity. The SSHSM model MUST provide mutual authentication of the client and server, and MUST authenticate, integrity-check, and encrypt the messages.

Upon establishment of a SSH session, the TMSP will cache the state information about the transport parameters. The tmSessionReference will be passed to the corresponding MPSP.

[todo] The tmSessionReference cache for use with the SSH Authentication Protocol [SSHAUTH]:

- tmStateReference
- tmSecurityStateReference
- tmTransportDomain = TCP/IPv4
- tmTransportAddress = x.x.x.x:y
- tmSecurityModel = SSHSM
- tmSecurityLevel = "authPriv"
- tmSessionID = Handshake session identifier
- tmSessionKey = Handshake peer certificate
- tmSessionMasterSecret = master secret
- tmSessionParameters = compression method, cipher spec, is-resumable

  tmSecurityName = "dbharrington"
  tmAuthMechanism = "[todo]"
  tmAuthProtocol = ""
  tmRadiusServer = ""
  tmPrivProtocol = ""

Additional information will be added to the tmStateReference by the authentication portion of the SSHSM.
4.3.1.1 Authenticating Servers and Clients

[todo] can we mandate mutual authentication?

4.3.2 [todo] Using Local Accounts to Authenticate Users

Upon creation of a SSH session leveraging SSH Local Accounts, the TMSP will cache the session authentication information in the tmSessionReference:

- tmSecurityName = "dbharrington"
- tmAuthMechanism = "public key"
- tmAuthProtocol = LocalAccounts

4.3.3 [todo] Using RADIUS Accounts to Authenticate Users

Upon creation of a SSH session leveraging RADIUS Accounts, the TMSP will cache the session authentication information in the tmSessionReference:

- tmSecurityName is the name used in username field of the RADIUS ACCESS-REQUEST message.
- tmAuthMechanism = "[todo]"
- tmAuthProtocol = RADIUS
- tmRadiusServer = x.x.x.x:y

4.3.4 securityStateReference for SSHSM

[todo]

- messageProcessingModel = SNMPv3
- securityModel = SSHSM
- securityName = tmSecurityName
- securityLevel = msgSecurityLevel

4.4 MIB Module for SSH Security Model

Each security model should use its own MIB module, rather than utilizing the USM MIB, to eliminate dependencies on a model that could be replaced some day. See RFC 3411 section 4.1.1.

The SSHSM-MIB module needs to provide the mapping from model-specific identity to a model independent securityName, and possibly a mapping to a groupname.

[todo] Module needs to be worked out once things become stable...

4.5 [todo] Notifications

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

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

5. Elements of Procedure

5.1 Establishing a Session

The Secure Shell Security Model provides the following internal primitive to pass data back and forth between the Security Model itself and the SSH authentication service:

```plaintext
statusInformation =
establishSession(
  IN transportDomain            -- transport domain to be used
  IN transportAddress          -- transport address to be used
  IN securityModel             -- Security Model to use
  IN securityEngineID        -- SNMP entity
  IN securityName              -- on behalf of this principal
  IN securityLevel             -- Level of Security requested
  OUT sessionID
)
```

The following describes the procedure to follow to establish a session between a client and sever to run SNMP over SSH. This process is followed by any SNMP engine establishing a session for subsequent use. In practice, this is done by an application that initiates a transaction, such as a Command Generator or a Notification Originator or a Proxy Forwarder. It is never triggered by an application preparing a response message, such as a Command Responder or Notification Receiver, because securityStateReference will always have session information for a response message.

The parameters necessary to establish a session are provided by the Secure Shell Security Model to the SSH client code, using the establishSession() ASI.

1) If the securityLevel specifies that the message is to be authenticated, but the SSH implementation does not support an authentication protocol, then the message cannot be sent. An error indication (unsupportedSecurityLevel) is returned to the calling module.

2) If the securityLevel specifies that the message is to be protected from disclosure, but the SSH implementation does not support
encryption, then the message cannot be sent. An error indication (unsupportedSecurityLevel) is returned to the calling module.

3) Using destTransportDomain and destTransportAddress, the client will establish an SSH transport connection using the SSH transport protocol, and the client and server will mutually authenticate, and exchange keys for message integrity and encryption. If the attempt to establish a connection is successful, then tmStateReference is created, and the values of destTransportDomain and destTransportAddress are saved. If the attempt to establish a connection is unsuccessful, then an error indication [todo] will be returned, and [todo] processing stops.

4) The provided securityEngineID is used to lookup the associated entry in the Local Configuration Datastore (LCD), and the securityName, securityModel, and securityLevel, information concerning the user at the destination is extracted. This step allows preconfiguration of model-specific user identities mapped to a securityName. Set the username in the SSH_MSG_USERAUTH_REQUEST to the username extracted from the LCD.

If information about the user is absent from the LCD, then set the username in the SSH_MSG_USERAUTH_REQUEST to the value of securityName. This allows a deployment without preconfigured mappings between model-specific and model-independent names, but the securityName will need to contain a username recognized by the authentication mechanism.

5) The client will then invoke the "ssh-userauth" service to authenticate the user, as described in the SSH authentication protocol [SSHAuth].

6) If the authentication is unsuccessful, then the transport connection should be closed, tmStateReference is discarded, the message is discarded, an error indication (unknownSecurityName) is returned to the calling module, and processing stops for this message.

7) Once the user has been successfully authenticated, the client will invoke the "ssh-connection" service, also known as the SSH connection protocol [SSHConnect].

8) After the ssh-connection service is established, the client will use an SSH_MSG_CHANNEL_OPEN message to open a channel of type "session", providing a selected sender channel number, and a maximum packet size based on maxMessageSize.

9) If successful, this will result in an SSH session. The
In order to allow SNMP traffic to be easily identified and filtered by firewalls and other network devices, servers associated with SNMP entities using the Secure Shell Security Model MUST default to providing access to the "SNMP" SSH subsystem only when the SSH session is established using the IANA-assigned TCP port (TBD). Servers SHOULD be configurable to allow access to the SNMP SSH subsystem over other ports.

todo] check whether there is a better way to establish a tunnel for SNMP messages.

todo] Should we perform some type of engineID discovery to provide the mapping between transport address, session, and engineID at this point in the session establishment procedure? We have an established channel; can we simply send a GET of snmpEngineID and record the value in the tmStateReference?

11) todo] the engine will perform an SNMP GET command requesting the value of the remote engine’s snmpEngineID object, and create a tmSessionReference cache recording the following information:
   - the remote engine’s snmpEngineID
   - the transport address
   - the recipient and sender channels

5.2 Discovery

Since snmpEngineID isn’t really needed for authentication and integrity checking, it becomes useful primarily for contextEngineID. contextEngineID is useful for proxy, and for a management application to uniquely identify an SNMP entity. Since snmpEngineID is an object in the SNMP-FRAMEWORK-MIB, the mapping between engineID and transport address could be established after a tunnel is established, or could be determined using noAuthNoPriv (with suitable caveats).

todo] Auto-discovery of SNMP devices is an important feature of many NMS platforms. Should we simply use a noAuthNoPriv request, and recommend an associated access control configuration that only makes
accessible relatively benign data such as sysOID, sysDescription, and snmpEngineID? Should we standardize this approach for all TMSM models, including a "named policy" for what can be discovered (a policy to be configured within whatever access control system is used)?

Alternatively, can we let USM perform discovery so we don’t have to attempt to establish an SSH connection first? USM is the mandatory-to-implement security model, so this could make sense.

5.3 Generating an Outgoing SNMP Message

This section describes the procedure followed by the Secure Shell Security Model whenever it generates a message containing a management operation (like a request, a response, a notification, or a report) on behalf of a user.

The parameters needed are supplied by the Message Processing Model via the generateRequestMsg() or the generateResponseMsg() ASI

statusInformation = -- success or errorIndication
generateRequestMsg(
  IN messageProcessingModel -- typically, SNMP version
  IN globalData -- message header, admin data
  IN maxMessageSize -- of the sending SNMP entity
  IN securityModel -- for the outgoing message
  IN securityEngineID -- authoritative SNMP entity
  IN securityName -- on behalf of this principal
  IN securityLevel -- Level of Security requested
  IN scopedPDU -- message (plaintext) payload
  OUT securityParameters -- filled in by Security Module
  OUT wholeMsg -- complete generated message
  OUT wholeMsgLength -- length of generated message
  OUT tmStateReference -- reference to session info
)
statusInformation = -- success or errorIndication
generateResponseMsg(
    IN messageProcessingModel  -- typically, SNMP version
    IN globalData              -- message header, admin data
    IN maxMessageSize          -- of the sending SNMP entity
    IN securityModel           -- for the outgoing message
    IN securityEngineID        -- authoritative SNMP entity
    IN securityName            -- on behalf of this principal
    IN securityLevel           -- Level of Security requested
    IN scopedPDU               -- message (plaintext) payload
    IN securityStateReference  -- reference to security state
    -- information from original
    -- request
    OUT securityParameters      -- filled in by Security Module
    OUT wholeMsg                -- complete generated message
    OUT wholeMsgLength          -- length of generated message
    OUT tmStateReference        -- reference to session info
)

verify securityModel = sshsmSecurityModel
If there is a securityStateReference, extract the tmStateReference
information from the cachedSecurityData from the Request message.
With USM, at the point the cachedSecurityData can now be
discarded. Since we now have persistent sessions, this is no
longer true, but can some of the cached data be discarded, such as
message pair information?
If there is no securityStateReference, then lookup the session
info indexed by (securityEngineID, securityName, securityLevel),
and set tmStateReference.
If there is no session info for this index, then create an
incomplete tmStateReference indexed by the provided
{securityEngineID, securityName, securityLevel}. Store the
securityModel and maxMessageSize information. When the TMSP gets
the incomplete tmStateReference, it will recognize that it needs
to establish a new session, and fill in the rest of the
information for subsequent use.
fill in securityParameters [todo] a NULL octet string since
    [todo] we don’t need to send securityEngineID, unless it is
    needed for a discovery mechanism..
    [todo] we don’t need to send Boots and Time values
    [todo] we don’t need to send a username, since we use the one
        from SSH authentication
    [todo] we don’t need to call authenticateOutgoingMsg()
The wholeMsg is now serialized and then represents the
unauthenticated message being prepared.
The completed message (wholeMsg) with its length (wholeMsgLength)
and securityParameters (a NULL octet string) and tmStateReference
is returned to the calling module with the statusInformation set
to success.

The Message Processing Model then passes information to the dispatcher for forwarding to the Transport Mapping.

5.4 Sending an Outgoing SNMP Message to the Network

The TMSP portion of the Secure Shell Security Model performs the following tasks:

- Uses tmStateReference to lookup session information.
- [todo] verifies that auth and priv can be provided, as requested, and error-out if not.
- If the session information is incomplete (i.e., has no tmTransportAddress), then call establishSession() using the destTransportDomain and destTransportAddress (the output of the PrepareOutgoingMessage() ASI) and the securityModel, securityEngineID, securityName, securityLevel from the tmStateReference. Store all information in the tmStateReference for subsequent use.
- An SSH_MSG_CHANNEL_DATA message is sent, indicating the recipient channel and encapsulating the wholeMessage.

[todo] According to RFC 3411, section 4.1.1, the application provides the transportDomain and transportAddress to the PDU dispatcher via the sendPDU() primitive. If we permit multiple sessions per transportAddress, then we would need to define how session identifiers get passed from the application to the PDU dispatcher (and then to the MP model).

[todo] The SNMP over TCP Transport Mapping document [RFC3430] says that TCP connections can be recreated dynamically or kept for future use and actually leaves all that to the transport mapping.

[todo] We might have a MIB module that records the session information for subsequent use by the applications and other subsytems, or it might be passed in the tmStateReference cache. For notifications, I assume the SNMPv3 notification tables would be a place to find the address, but I’m not sure how to identify the presumably-dynamic session identifiers. The MIB module could identify whether the session was initiated by the remote engine or initiated by the current engine, and possibly assigned a purpose (incoming request/response or outgoing notifications). First we need to decide whether to handle notifications and requests in one or two (or more) sessions. Do we use an established session bi-directionally, or do we establish two separate sessions, one for each direction as needed?
5.5 [todo] Prepare Data Elements from an Incoming SNMP Message

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

5.6 Processing an Incoming SNMP Message

This section describes the procedure followed by an SNMP engine whenever it receives a message containing a management operation on behalf of a user.

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. Also, an error indication can return an OID and value for an incremented counter and optionally a value for securityLevel, and values for contextEngineID or contextName for the counter. In addition, the securityStateReference data is returned if any such information is available at the point where the error is detected. [todo] this paragraph may no longer be accurate because of persistent session state information.

The abstract service primitive from a Message Processing Model to the Security Subsystem for a received message is:

\[
\text{statusInformation} = \begin{cases} 
\text{errorIndication or success} \\
\text{error counter OID/value if error}
\end{cases}
\]

\[
\text{processIncomingMsg}( \\
\text{IN messageProcessingModel -- typically, SNMP version} \\
\text{IN maxMessageSize -- of the sending SNMP entity} \\
\text{IN securityParameters -- for the received message} \\
\text{IN securityModel -- for the received message} \\
\text{IN securityLevel -- Level of Security} \\
\text{IN wholeMsg -- as received on the wire} \\
\text{IN wholeMsgLength -- length as received on the wire} \\
\text{OUT securityEngineID -- authoritative SNMP entity} \\
\text{OUT securityName -- identification of the principal} \\
\text{OUT scopedPDU, -- message (plaintext) payload} \\
\text{OUT maxSizeResponseScopedPDU -- maximum size sender can handle} \\
\text{OUT securityStateReference -- reference to security state}
\end{cases}
\]

1) If the received securityParameters is not the serialization of an OCTET STRING formatted according to the SSHsmSecurityParameters,
then the snmpInASNParseErrs counter [RFC3418] is incremented, and an
counter, which may be important if this security model
error indication (parseError) is returned to the calling module.
	[todo] Note that we return without the OID and value of the

todo] If we actually do not extract anything from
error indication (parseError) is returned to the calling module.  
securityParameters, do we need to check whether this parses
correctly?  It apparently parsed well enough to pass the parse test
in the messaging model.  Could we simply ignore the
securityParameters being passed in?  The only argument I see for
checking to ensure this is empty/null is to ensure somebody isn’t
using the filed for non-standard purposes, such as passing a virus in
the field.

2) The SSHSM queries the associated SSH engine, in an
implementation-dependent manner, to determine the transport and
security parameters for the received message:
   a) the transportDomain and transportAddress
   b) the authentication parameters, including the authenticated
      username
   c) the encryption options,
   d) the integrity-checking options

3) The securityEngineID to be returned to the caller is determined in
an implementation-dependent manner, such as by using the transport
address to perform a lookup in its Local Configuration Datastore
(LCD).  If the securityEngineID is unknown, then an SNMP engine may
perform discovery to create a new entry in its LCD and continue
processing.  Note that securityEngineID is required by the SNMPv3
message processing model in RFC 3412 section 7.2 13a)

4) If the information about the message security indicates that the
security options do not match the securityLevel requested by the
caller, then the SSHsmStatsUnsupportedSecLevels counter is
incremented and an error indication (unsupportedSecurityLevel)
together with the OID and value of the incremented counter is
returned to the calling module.

5) The scopedPDU component is assumed to be in plain text and is the
message payload to be returned to the calling module.

7) The maxSizeResponseScopedPDU is calculated.  This is the maximum
size allowed for a scopedPDU for a possible Response message.
Provision is made for a message header that allows the same
securityLevel as the received Request.
[todo] Is this relevant? Can it be calculated once for the session? Do we need to take into consideration the SSH window size?

10) Information about the value of the SSH username is extracted from the Local Configuration Datastore (LCD) to provide conversion from the SSHSM model-specific username to a model-independent securityName. If no information is available for the username, then the securityName is set to the username used in the SSH-USER-AUTH-REQUEST. [todo] Note that USM at this point would flag an unknownSecurityName error.

11) The security data is cached as cachedSecurityData, so that a possible response to this message can and will use the same authentication and privacy parameters. Information to be saved/cached is as follows:
   - transportDomain, transportAddress
   - securityEngineID
   - SSH username, [todo]
   - auth options [todo]
   - encryption options [todo]
   - Integrity checking options [todo]

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

6. MIB module definition

SSHSM-MIB DEFINITIONS ::= BEGIN

IMPORTS
   MODULE-IDENTITY, OBJECT-TYPE,
   OBJECT-IDENTITY, mib-2 FROM SNMPv2-SMI
   TEXTUAL-CONVENTION FROM SNMPv2-TC
   MODULE-COMPLIANCE, OBJECT-GROUP FROM SNMPv2-CONF
   SnmpSecurityModel FROM SNMP-FRAMEWORK-MIB;

sshsMIB MODULE-IDENTITY
   LAST-UPDATED "200509020000Z"
   ORGANIZATION "ISMS Working Group"
   CONTACT-INFO "WG-EMail: isms@lists.ietf.org
                  Subscribe: isms-request@lists.ietf.org"

Chair:
   Juergen Quittek
   NEC Europe Ltd.
   Network Laboratories
   Kurfuersten-Anlage 36


**DESCRIPTION**  "The Secure Shell Security Model MIB

Copyright (C) The Internet Society (2005). 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

```
::= { mib-2 xxxx }
-- RFC Ed.: replace xxxx with IANA-assigned number and
-- remove this note
```
-- subtrees in the SSHSM-MIB

sshsmNotifications OBJECT IDENTIFIER ::= { sshsmMIB 0 }
sshsmObjects OBJECT IDENTIFIER ::= { sshsmMIB 1 }
sshsmConformance OBJECT IDENTIFIER ::= { sshsmMIB 2 }

-- Objects

-- sshsmMIB - Conformance Information

sshsmGroups OBJECT IDENTIFIER ::= { sshsmConformance 1 }
sshsmCompliances OBJECT IDENTIFIER ::= { sshsmConformance 2 }

-- Units of conformance

sshsmGroup OBJECT-GROUP
    OBJECTS {
    }
    STATUS current
    DESCRIPTION "Secure Shell Security Model objects"
    ::= { sshsmGroups 2 }

-- Compliance statements

sshsmCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION "The compliance statement for support of the Secure Shell Security Model"
    MODULE
    MANDATORY-GROUPS {
    }
    ::= { sshsmCompliances 1 }


7. Security Considerations

This document describes a security model that would permit SNMP to utilize SSH security services. [todo] expand as needed.

SSHv2 provides PFS for encryption keys. PFS is a major design goal of SSH, and any well-designed keyex algorithm will provide it.

[todo] We will probably need to discuss the security implications of password based authentication methods.

8. IANA Considerations

IANA is requested to assign:
  a TCP port number which will be the default port for SNMP over SSH sessions as defined in this document,
  an SMI number under mib-2, for the MIB module in this document,
  an SNMP SecurityModel for the Secure Shell Security Model, as documented in the MIB module in this document,

9. Acknowledgments

The editors would like to thank Jeffrey Hutzelman and Nicholas Williams for sharing their SSH insights.

10. References

10.1 Normative References


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


10.2 Informative References


Authors’ Addresses

David Harrington  
Effective Software  
Harding Rd  
Portsmouth NH  
USA  
Phone: +1 603 436 8634  
Email: dbharrington@comcast.net

Juergen Schoenwaelder  
International University Bremen  
Campus Ring 1  
28725 Bremen  
Germany  
Phone: +49 421 200-3587  
Email: j.schoenwaelder@iu-bremen.de
Appendix A. Change Log from the first revision of -00-

-00-b draft:
re-ordered the sections from abstract to concrete.
worked on how the SSHSM fits into the RFC3411 and TMSM architectures.
added goals to the Motivation section.
worked on Security Capabilities based on input from Joe, Nick, and JHutz. Added Joe to the authors list based on contributed text.
created Data origin Authentication section, to separate this discussion.
expanded "Authentication Protocol" section
Updated Message replay section.
--00-c draft
worked on security information cacheing, including breaking caches into session, message, and model-independent (which we probably want to remerge later)
eliminated a lot of TMSM-carryover stuff and modified to be SSHSM-specific.
updated references.
filled in Elements of Procedure for Outgoing Messages
created shell for SSHSM-MIB
-01- draft
added Processing an Incoming Message
updated change log
modified masquerade discussion to differentiate server vs user authentication
added [todos] for Data origin Authentication Issues section, trying to nail down whether we even need this section.
disallow the ‘none’ MAC algorithm
added text to Message security versus session security
wordsmithed "authentication protocol" section
rewrote "Mapping SSH to EngineID", eliminating most [todos]
modified "Establishing a Session"
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to 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.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

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