

Standard Diagnostic Logging
Version 1.0
13 January 2016

This is a Binding Permanent Reference Document of the GSMA

Security Classification: Non-confidential

Access to and distribution of this document is restricted to the persons permitted by the security classification. This document is confidential to the Association and is subject to copyright protection. This document is to be used only for the purposes for which it has been supplied and information contained in it must not be disclosed or in any other way made available, in whole or in part, to persons other than those permitted under the security classification without the prior written approval of the Association.

Copyright Notice

Copyright © 2016 GSM Association

Disclaimer

The GSM Association ("Association") makes no representation, warranty or undertaking (express or implied) with respect to and does not accept any responsibility for, and hereby disclaims liability for the accuracy or completeness or timeliness of the information contained in this document. The information contained in this document may be subject to change without prior notice.

Antitrust Notice

The information contain herein is in full compliance with the GSM Association's antitrust compliance policy.

Table of Contents

1 Introduction 3
1.1 Overview 3
1.2 Scope 4
1.3 Definitions 4
1.4 Abbreviations 5
1.5 References 5
2 Standard Diagnostic Interface Overview 6
3 Cross Platform Compatibility 7
4 Use of SNMP 8
4.1 Method of SNMP Message Exchange 9
5 MDMI 9
5.1 Required Functions 9
5.1.1 MdmiCreateSession 9
5.1.2 MdmiCloseSession 9
5.1.3 MdmiGet 10
5.1.4 MdmiSet 10
5.1.5 MdmiInvoke 11
5.1.6 MdmiSetEventCallback 11
5.1.7 MdmiSubscribe 12
5.1.8 MdmiUnsubscribe 12
5.2 MIB 12
6 Security 13
6.1 Example of a JSON CommandToken 14
6.1.1 Steps for Building the Token 15
6.1.2 Steps for Parsing the Token 15
6.2 Log Security 15
6.2.1 Introduction 15
6.2.2 Log Mask Management 15
Annex A Source Code 17
Annex B Document Management 18
B.1 Document History 18
Other Information 18

1 Introduction

1.1 Overview

The purpose of this document is to provide a standardized method to log modem data and messaging on a device, eliminating the need for tethered logging. The primary user of the logging tool is expected to be mobile network operators.

Possible use cases are listed below:

- Report the geo-location of the device and key RF parameters (RSRP, RSSI, SINR etc) to determine network coverage
- Present geo-located events on maps to allow better call drop analysis
- Capture handover statistics to debug handover issues
- Report VoLTE (Voice over LTE) call statistics (e.g. Delay, Jitter and Packet Loss) to aid in VoLTE analysis
- Real time reporting on the device
    - Data throughput
    - Txpower
    - Cell selection
    - RF parameters

The operator will be able to log with any device/chipset compliant to the interface outlined in this document by downloading a compliant application. The logs will be saved on the device and uploaded to an operator server. This document provides the APIs (Application Programming Interface) and MIB (Management Information Base) to capture the modem data, and the security protocol to authenticate the device before logging can be initiated. The API source code is available on the GSMA GitHub site, the manufacturers and application vendors are encouraged to download this source code to implement the standard logging solution.

The API defined in this document to retrieve the modem data are part of the MDMI (Modem Diagnostic Monitoring Interface). Using the MDMI, the diagnostic application will be able to retrieve information from the modem, such as a KPI or a protocol message. MDMI is defined using a standardized format, SNMP (Simple Network Management Protocol). As per SNMP convention, all information retrieved from the modem is passed as objects and are defined in the MIB. Some advantages of using this framework to define the interface to the modem.

- Since SNMP is operating system agnostic, the MDMI interface can be used by any OS.
- As more functionality is requested, additional KPIs, protocol and commands can be easily added

This document is organized as follows:

Section 2 introduces the MDMI (Modem Diagnostic Monitoring Interface) and presents an overview of how it utilizes SNMP (Simple Network Management Protocol)

Section 3 presents architectures on the device to implement the standard diagnostic interface

Section 4 describes how SNMP has been adapted for use in MDMI, including an overview of the types of information that MDMI makes available, the relationships between the different application components, and method that is used for exchanging messages.

Section 5 outlines MDMI, including the functions that must be implemented, and provides an overview of the MIB (Management Information Base), which defines all the messages, KPIs (Key Performance Indicators) and commands that must be made available.

Section 6 describes the authentication of the device required before logging can take place.

Annex A references the source code location.

1.2 Scope

The initial scope of the GSMA standard diagnostic logging interface are restricted to engineering builds for LTE and Wi-Fi only. Expansions of the scope will be possible after further study.

1.3 Definitions

Term	Description
ASN.1 BER	The encoding used to pack a Log Record, as in SNMP
DM App	Diagnostic Monitoring Application - any app that uses MDMI
Event	A Log Record pushed by MDMI to a DM App
KPI	Key Performance Indicator - KPI Log Records report KPIs
Log Record	A single piece of diagnostic information from MDMI, either pulled from MDMI by a DM App, or pushed by MDMI to the DM App
MDMI	Modem Diagnostic Monitoring Interface - interface defined by this document
MDMI Session	An identifier MDMI uses to identify a particularDM App - this will be assigned during MDMI initialization
MDMI Value	The contents of a Log Record - either KPIs or a protocol message - encoded using ASN.1 BER into a buffer of bytes
MIB	Management Information Base – set of diagnostic objects that can be managed by SNMP
OID	Object ID – the unique identifier for an SNMP object
Protocol Message	A Log Record containing an OTA message of a particular protocol
SNMP	Simple Network Management Protocol
SNMP Agent	The provider of diagnostic information - in MDMI, the OEM writes a library conforming to this requirement that acts as an SNMP agent
SNMP Manager	The user of SNMP Agent - in MDMI, this would be a component of a DM app
SNMP Object	A unit of diagnostic information defined in the MIB
SNMP Trap	SNMP terminology for a message pushed from agent to manager - in MDMI, we use the term "Event" synonymously

1.4 Abbreviations

Term	Description
API	Application Programming Interface
DM App	Diagnostic Monitoring Application
KPI	Key Performance Indicator
MDMI	Modem Diagnostic Monitoring Interface
MIB	Management Information Base
OID	Object ID
SNMP	Simple Network Management Protocol

1.5 References

Several standards were used to develop MDMI. They are listed here as a reference.

References	
SNMP	
SNMP standards were used to define the MIB and messaging format.	
Structure and Identification of Management Information	http://www.ietf.org/rfc/rfc1155.txt
SNMP	http://www.ietf.org/rfc/rfc1157.txt
MIB	http://www.ietf.org/rfc/rfc1212.txt
MIB-2	http://www.ietf.org/rfc/rfc1213.txt
SNMP Traps	http://www.ietf.org/rfc/rfc1215.txt
3GPP	
All logged messages should be reported in their original format without modification, as described by the 3GPP standard.	
RRC	http://www.3gpp.org/DynaReport/36331.htm
NAS	http://www.3gpp.org/DynaReport/24301.htm
MAC	http://www.3gpp.org/DynaReport/36321.htm
PDCP	http://www.3gpp.org/DynaReport/36323.htm
RLC	http://www.3gpp.org/DynaReport/36322.htm
IP	
IP Packets (and all protocol messages contained therein) logged should also be in their original format without modification, as described by IETF.	
IPv4	http://www.ietf.org/rfc/rfc791.txt
IPv6	http://www.ietf.org/rfc/rfc2460.txt

2 Standard Diagnostic Interface Overview

The standard logging interface is referred to in this document as the MDMI, which is an application programming interface (API) and a messaging interface between Diagnostic Monitoring (DM) applications running on a mobile device and the device modem. MDMI enables DM applications to monitor and control the activities of the device modem. The messaging interface is based on a modified version of the Simple Network Management Protocol (SNMP).

outlines an example use case of primary functions of this interface, and how SNMP is used to achieve this function.

Function	Example	Mechanism
Interrogate	What is the device's current RSRP?	SNMP GetRequest
Configure	Set Airplane Mode On	SNMP SetRequest
Command	Make a telephone call	SNMP SetRequest
Log Subscribe	Subscribe to all RRC Messages	Specify Mask: SNMP SetRequest Receive Information: Event (SNMP Trap)

Table 1 Primary Functionality

The information that can be retrieved through MDMI can be described as SNMP Log Records. Types of Log Records are listed in

.

Type	Examples	Pushed/Pulled	Format Definition
KPI Log Record	• RSRP • PUSCH Tx Power • Path Loss	Push or Pull	Defined in MIB
Protocol Message Log Record	• RRC Messages • NAS Messages • IP Packets	Push	• Header defined in MIB • Payload defined by relevant standard
Command Result Log Record	• Success of Phone Call • Location determined by Fix	Push	Defined in MIB
Configuration Log Record	• Device Name • MDMI Version	Pull	Defined in MIB

Table 2 Log Record Types

As illustrated in Figure 1, the device OEMs are expected to implement an SNMP agent that provides the MDMI programming interface specified in this document. DM applications developed by third parties utilize the MDMI programming interface to monitor and control the device modem. When invoked by a DM application using MDMI, the SNMP Agent in the application processor (A-processor) could interact with the modem in the B-processor via inter-process communication (IPC) to perform the monitoring and control functions requested by the DM application. MDMI is generically defined with the goal that it is implementable by an OEM, regardless of the device operating system or the device chipset. As such, a DM application using MDMI should run on any such device without modification.

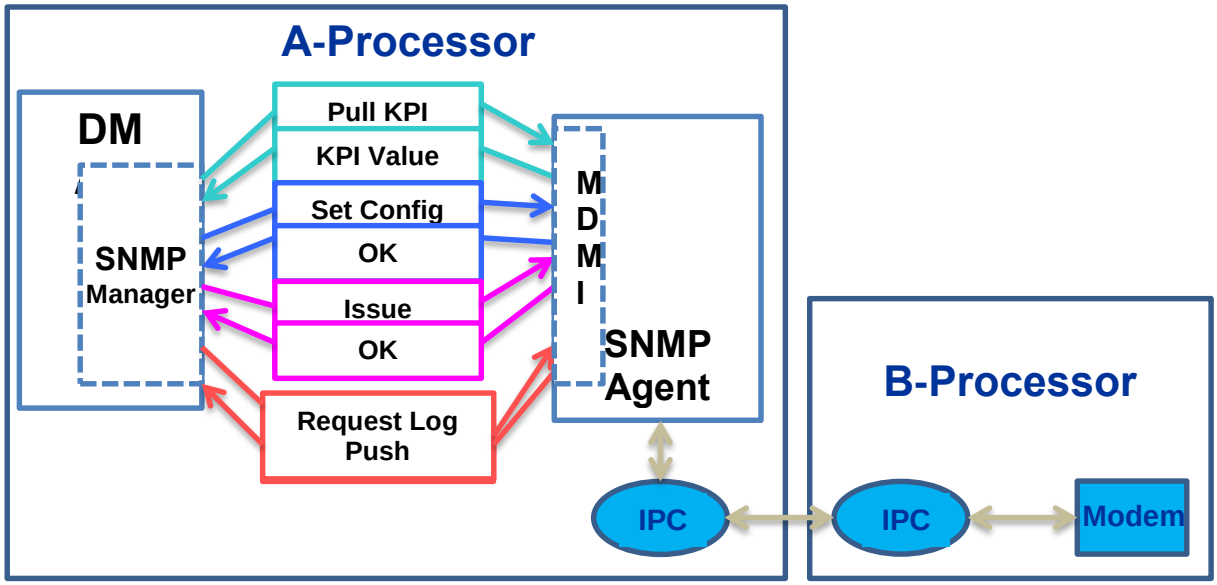
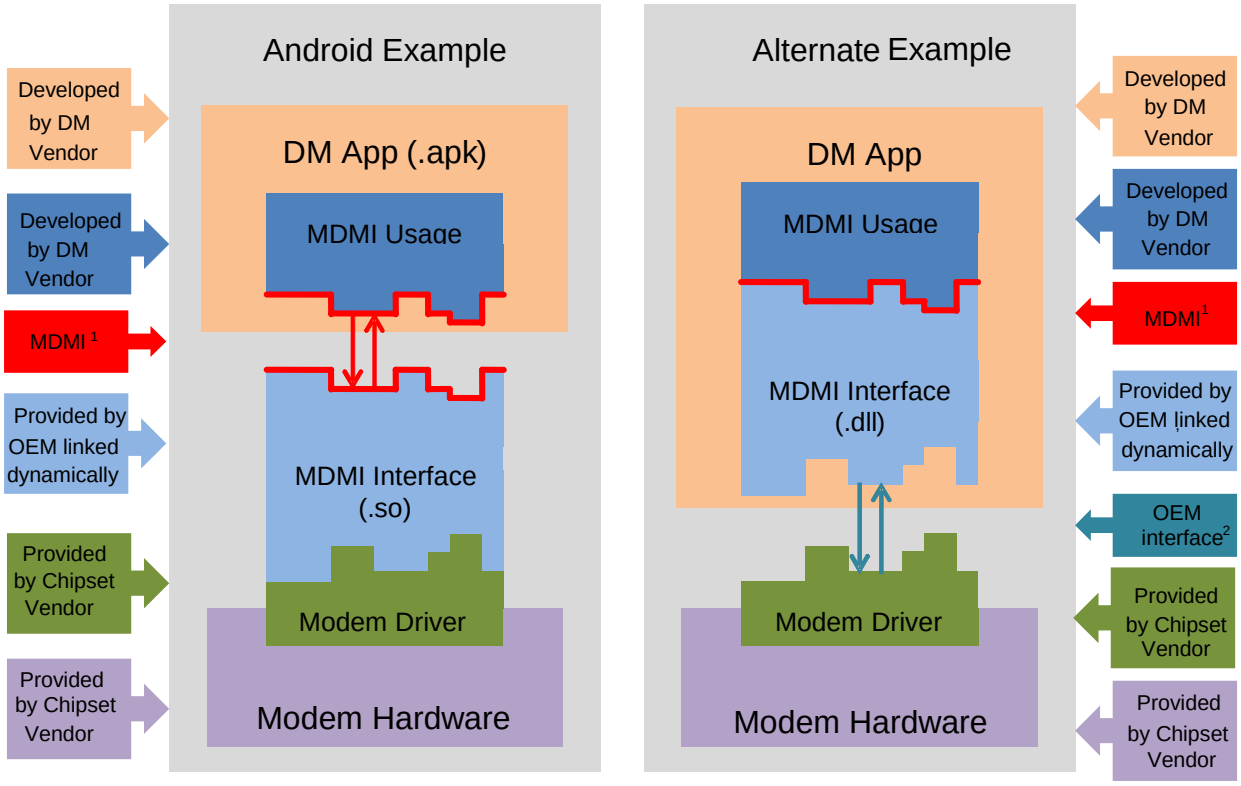


Figure 1 Design Architecture

3 Cross Platform Compatibility

This API library has been designed to make integrating aDM App onto devices as seamless as possible. The usage should be identical across devices, chipsets, OEMs and even operating systems. However, due to underlying differences between current mobile operating systems, some variance will be inevitable.

As illustrated in **Error! Reference source not found.**, the library may be provided differently on Android from how it is provided on operating systems with platform restrictions. In both cases, the OEM will provide a library of executable functions, making all the functionality of MDMI available to a DM app. On Android, the library will be pre-installed on the system by the OEM. A DM app, once installed, will dynamically link to this library to use the functionality of MDMI. On an OS with platform restrictions, pre-installing the library is not possible. The library will be prepared by the OEM and provided to the developer of the DM App. This library will be compiled directly into the DM app. TheDM App will have to be recompiled for each version of the library.



[1]This interface is defined by MDMI, and must conform to the definition
[2]This interface is not defined by MDMI, and is up to the OEMs to implement

Figure 2 MDMI Platform Comparison

4 Use of SNMP

The MDMI interface is based on SNMP, in spite of their differences (notably, that the interface is not over UDP/IP). The reason for basing the interface on SNMP is to make use of a standardized monitoring and control interface structure. As described in Section 3, the DM application includes the functionality of a SNMP Manager, and the MDMI module, which provides a programming interface to extract modem information, acts as a SNMP Agent.

The desired implementation can be achieved with SNMPv1 and, only a subset of SNMPv1 is used.

The MIB defines all SNMP objects that are available through MDMI. These are organized hierarchically, and each object has an OID as identifier. Some are available to be read, and others can be both read or written. Some are available to be pushed to theDM App as events once theDM App has subscribed to them. The MIB also defines the exact syntax of each field in each object. Initially the scope of these objects covers a set of LTE and Wi-Fi KPIs and protocol messages, as well as some basic configuration items and commands. The design is extensible so that more KPIs, protocols and commands can be added in future releases.

4.1 Method of SNMP Message Exchange

SNMP operates over UDP. MDMI replaces this with function calls in which a buffer is passed from theDM App to the SNMP Agent and vice versa. This buffer will contain the SNMP requests/responses encoded in the same way SNMP is encoded (ASN.1 BER encoding scheme, see references in section 1 for more details). Both open source and commercial libraries exist to encode and decode ASN.1, and are widely used in many other telecommunication protocols.

An application can either pull logs from MDMI by using a "Get" message, or the application can request that logs to be pushed to it by specifying which events should be sent from the MDMI to the DM App. When events are to be pushed, theDM App must specify the function MDMI will use as a pointer using `MdmiSetEventCallback`, and specifying the events that are requested using `MdmiSubscribe`.

5 MDMI

The basics of the API are described below. Devices implementing MDMI must conform to these definitions precisely.

5.1 Required Functions

Each required function is listed in the tables below with:

- Function Name - the name of the function
- Signature - the exact function signature that must be implemented
- Arguments - name and explanation of the arguments passed to the function
- Return Value - value returned by the function

5.1.1 MdmiCreateSession

This function creates a MDMI session that is used in subsequent MDMI calls to identify the caller.

Function Name	**MdmiCreateSession**

Signature	`MdmiError` **`MdmiCreateSession`**
	`(const wchar_t* address,`
	`MdmiSession* session);`

Arguments	`addess`: address of the MDMI device to open. May be ignored if the system has only one device `session`: session object that will be set upon success. This will be used by the caller in subsequent calls to MDMI.
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.2 MdmiCloseSession

This function closes the MDMI session.

Function Name	**MdmiCloseSession**
Signature	`MdmiError` **`MdmiCloseSession`** `(MdmiSession* session);`
Arguments	`session`: session object that will be closed. If close is succesful, the session object is set to 0, indicating invalid session
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.3 MdmiGet

This function gets the value of a specific object, as specified by that object's OID.

Function Name	**MdmiGet**

Signature	`MdmiError` **`MdmiGet`**
	`(MdmiSession session,`
	`const MdmiObjectName* name,`
	`MdmiValue* value);`

Arguments	`session:` identifies the session `name:` OID of the value `value:` value to be read. If the read is succesful, the actual value is read into this pointer. Upon return the ownership of this pointer is transferred to caller and must be freed when no longer needed
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.4 MdmiSet

This function sets the value of a specific object, as specified by that object's OID.

Function Name	**MdmiSet**

Signature	`MdmiError` **`MdmiSet`**
	`(MdmiSession session,`
	`const MdmiObjectName* name,`
	`const MdmiValue* value);`

Arguments	`session:` identifies the session `name:` OID of the value `value:` value to be set
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.5 MdmiInvoke

This function invokes a command through MDMI. Commands are defined in the MIB and identified by an OID.

Function Name	**MdmiInvoke**

Signature	`MdmiError` **`MdmiInvoke`**
	`(MdmiSession session,`
	`const MdmiObjectName* name,`
	`const MdmiValue* value);`

Arguments	`session:` identifies the session `name:` OID of the command to invoke `value:` optional value of the command (can be null)
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.6 MdmiSetEventCallback

This function sets the callback function that will be used for pushed events.

Function Name	**MdmiSetEventCallback**

Signature	`MdmiError` **`MdmiSetEventCallback`**
	`(MdmiSession session,`
	`MdmiEventCallback callback,`
	`void* state);`

Arguments	`session:` identifies the session `callback:` The callback function pointer. This value will replace previous value. Setting this value to NULL will stop event callbacks. See MDMI.h for definition of the callback. `state:` Optional state that will be passed when callback function is called
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.7 MdmiSubscribe

This function specifies an object, which should be reported via trap message whenever it is updated.

Function Name	**MdmiSubscribe**

Signature	`MdmiError` **`MdmiSubscribe`**
	`(MdmiSession session,`
	`const MdmiObjectName* name);`

Arguments	`session:` identifies the session `eventName:` identifies the event to be registered. Multiple registrations will still result in only one event being generated
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.8 MdmiUnsubscribe

This function removes an object from the list of objects that should be reported via trap message.

Function Name	**MdmiUnsubscribe**

Signature	`MdmiError` **`MdmiUnsubscribe`**
	`(MdmiSession session,`
	`const MdmiObjectName* name);`

Arguments	`session:` identifies the session `eventName:` identifies the event to be deregistered
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.2 MIB

The MIB is designed to be compliant with SNMPv1 (see Reference Section 1). It provides a set of objects which can be retrieved and set through the MDMI.

The MIB contains the following five sections:

1. Configuration - device and MDMI configuration information
2. KPIs - statistics which can be monitored to determine modem behaviour
3. Messages - OTA traffic between the modem and the network
4. Commands - commands that MDMI makes available
5. Traps & Trap Subscriptions - allows theDM App to specify which Log Records should be pushed back to theDM App as events via SNMP Traps

6 Security

The operator owned server manages both authentication and log sessions in devices. User is authenticated prior to a session and a command token is then sent from server to the device.

The token is passed to the modem by the diagnostic application, where the integrity of the token is verified and then parsed.

Selective logging, based on log mask setting in the command token, is enabled. Logs are collected by the diagnostic application, prior to sending them to the operation server.

Note: MDMI API for log session control functions, such as log mask setting, is used for development phase of the device and diagnostic application.

1. **User Authentication** – Server validates user with login credentials (user name and password), as supplied by user launching the application, thereby restricting only authenticated users to log into the diagnostic system. Additional checks based on IMEI and user's group are used to restrict devices log and reports to specific users or user group.
2. **Secure Channel** – Command tokens are sent to the device from the server via a secure channel
3. **Command Set** – JSON packets are used to define command tokens (ON/OFF, Time to Live, Log Privilege level, Log groups). JSON Packets are integrity protected and can be interpreted only by the modem (which has the public key of policy certificate).
4. **Log Data** – Protection based on log mask. Log mask can be set by user or priviliges enforced by the SW load.

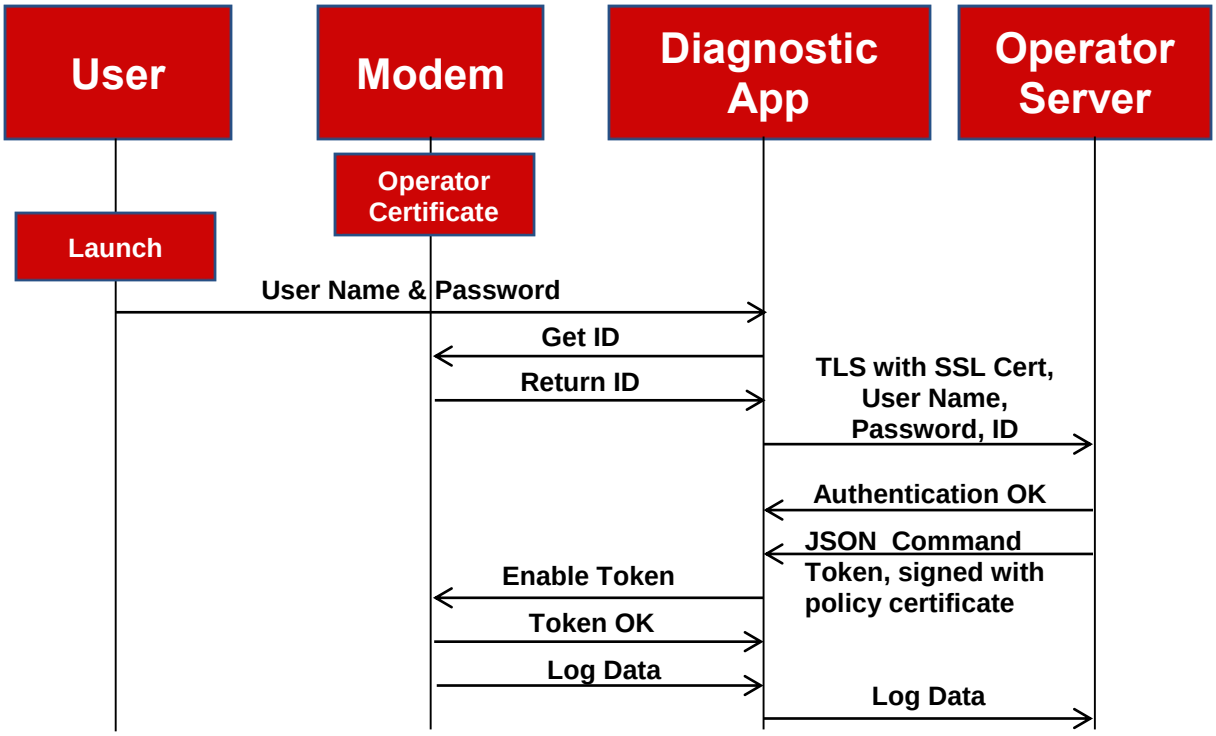


Figure 3 Security Call Flow

6.1 Example of a JSON CommandToken

An example of a command token sent from operator server to the device, is provided below:

- "token_id", identifier for a token for the operator server
- "session_id", identifier for a particular log session
- "device_id", IMEI of the device
- "validity", length of log session
- "diag_cmd",diagnostic command
- "diag_mask", diagnostic log mask, specify what needs to be logged. Example, all RRC messages.

{

"diag_token":["token_id":"0xaf010230405", "session_id" : "0x1234435",
"device_id":"990000862471854","validity" : "8/6/2015 5:00:00PM EST",
"diag_cmd":"start_log", "diag_mask":"dnp3cm9ja3M="],

"signature":"OGI0ZDM4NTY0MzMyYTVmYWI2OGRhNTMzMmJmNWY5MThhZjU5ZWViNQ==="

}

6.1.1 Steps for Building the Token

1. Server builds a Hash of the "diag_token" array, for integrity protection:
   SHA_1 (["token_id":"0xaf010230405", "session_id" : "0x1234435",
   "device_id":"990000862471854","validity" : "8/6/2015 5:00:00PM EST",
   "diag_cmd":"start_log", "diag_mask":"dnp3cm9ja3M="])

   =8b4d38564332a5fab68da5332bf5f918af59eeb5

2. Server signs the token using the private key of the policy certificate, for validity.
3. Server generates a base 64 encoded stream of the above token. Refer to "signature" object in above example.

6.1.2 Steps for Parsing the Token

1. Modem creates a Hash of the command token that it received.
2. Modem decodes the base 64 encoded value from the "signature" object.
3. Modem decrypts the binary signature above, using public key of the policy certificate. The Hash generated by server is obtained.
4. Modem compares the Hash from steps 3 and 1, to ensure integrity of the received token. If integrity is verified, Modem parses the command and enables logging.

Note: The Hashing algorithm and asymmetric encryption algorithm can be defined by the implementer. The implementer can enhance the steps by using industry standard algorithms such as HMAC.

6.2 Log Security

6.2.1 Introduction

At this time, since the scope of the logging per this document is restricted to engineering builds, encryption is not considered. In this initial phase, the log mask can be used to control the content of the log via user opt in/consent and privileges enforced by the SW load to address consumer privacy concerns. If user builds are supported in the future, encryption will need to be revisited at that time to prevent the log content from getting to unauthorized users.

6.2.2 Log Mask Management

The user/consumer payload, such as voice and emails, are most sensitive due to privacy issues. The best method to manage privacy concerns is an a priori setting, before logging starts. Log mask can be set such that all user payload (IP/UDP/TCP/RTP) are not logged, only the header information of these protocols will be logged.

Note that for SIP and RTCP, if packet logging is turned on, both header and payload needs to be logged. This is because the payload of SIP and RTCP includes session options which are used for debug and analysis. This should not be a concern since consumer data is not available in SIP and RTCP payloads.

The log mask setting can be managed by an application configuration/consent where a user is provided an option/interface to select what data can be logged. The log mask can also be enforced with the SW load. For example, the operator field engineers may receive a load where all log masks are available, but if 3rd party vendors are testing for the operator, they may get a load where certain log masks cannot be enabled.

The logmask choices can be either packet header only logging, or packet payload logging. If the user opts for packet header only logging, the user can further select any combination of the following:

1. IP header,
2. UDP header,
3. TCP header,
4. SIP header and payload,
5. RTP header,
6. RTCP header and payload

If the user opts for packet payload logging, the user can further select any combination of the following:

1. IP header and payload,
2. UDP header and payload,
3. TCP header and payload,
4. SIP header and payload,
5. RTP header and payload,
6. RTCP header and payload

Annex A Source Code

Open source code is publicly available for vendors who are complying with the GSMA TSG standard diagnostic interface standard at GSMA site:

https://github.com/GSMATerminals/TSG-Standard-Diag-Public.

Summary of the files available are listed below.

MDMI.h - header file specifying the MDMI interface.

MDMI-MIB.txt - MIB file specifying SNMP objects.

Sample.c - example source code demonstrating MDMI usage.

Note: A user account must be created at the following site before access to the GitHub site.

https://github.com/

Annex B Document Management

B.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	January 2016	New PRD TS.31	TSG#21 PSMC	Carol Becht / Verizon

Other Information

Type	Description
Document Owner	Terminal Steering Group
Editor / Company	Carol Becht / Verizon

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.