|  |  |
| --- | --- |
| Input Contribution | |
| Meeting ID\* | TST#28 |
| Title:\* | Details on PSK-based SAEF for the Security Developer’s Guide TR-0038 |
| Source:\* | Wolfgang Granzow, Qualcomm, [wgranzow@qti.qualcomm.com](mailto:wgranzow@qti.qualcomm.com)  Phil Hawkes, Qualcomm, [phawkes@qti.qualcomm.com](mailto:phawkes@qti.qualcomm.com) |
| Uploaded Date:\* | 2017-03-28 |
| Document(s)  Impacted\* | TR-0038v0\_1\_0 |
| Intended purpose of  document:\* | Decision  Discussion  Information  Other <specify> |
| Decision requested or recommendation:\* | Include the text proposed in this contribution into TR-0038. |
| Template Version:23 February 2015 (Do not modify) | |

**oneM2M Notice**

The document to which this cover statement is attached is submitted to oneM2M. Participation in, or attendance at, any activity of oneM2M, constitutes acceptance of and agreement to be bound by terms of the Working Procedures and the Partnership Agreement, including the Intellectual Property Rights (IPR) Principles Governing oneM2M Work found in Annex 1 of the Partnership Agreement.

**Introduction**

This contribution proposes additional details to the description of Provisioned Symmetric Key SAE agreed as clause 7.1 of TR-0038. These additional details are proposed to be included into an Annex.

R01: addition of Editor’s note at the end, correction of the openssl commands

If this contribution is agreed, the Editor’s note at the end of clause 7.1.2 is proposed to be removed:

*======== NEW Text proposed for TR-0038 starts here =================*

# Annex A:

# Security Association Establishment Message Flows

## A.1 Introduction

This Annex presents some example message flows which are useful to understand the operation of the oneM2M security establishment frameworks, to verify correct operation or to identify the cause of misbehavior.

Some details of TLS message flows and message content depend on the employed SSL/TLS implementation. Implementations of oneM2M entities will typically make use of SSL/TLS libraries to enable support of the required security functions specified in TS-0003. Examples of open source SSL/TLS libraries include *OpenSSL*, *gnuSSL* and *mbed TLS*.

Such SSL/TLS libraries implement the basic cryptographic functions and provides various utility functions such as e.g. TLS clients and servers which may be executed from a command line.

The message flows shown here have been produced using OpenSSL Version 1.1.1 on an Ubuntu 14.04 computer using the s\_client and s\_server utility functions, and employing Wireshark for capturing and analyzing the exchanged data packets.

The commands given in the subsections below may be used to reproduce these flows.

## A.1 PSK-Based Security Association Establishment

A typical flow of messages and actions for a successful PSK-Based Security Association Establishment is shown in figure A.1-1. The message content described in the steps below applies to the example described in clause 7.1.2.

Subsequent to TCP connection establishment (not shown in the Figure), the following messages are exchanged between ADN-AE1 and the MN-CSE:

1. The TLS client on ADN-AE1 sends a Client Hello Handshake message which is encapsulated in a TLS Record layer frame. The record layer message includes the following fields:
2. Record layer header fields:
   * + - Content type 0x16 (Handshake)
       - Version 0x0301 (indicating TLS 1.0)
       - Length of the message (2 bytes, value depending on the message content)
3. Application data (handshake message):
   * + - Handshake Type 0x01 (Client Hello)
       - Length of the message (3 bytes, value depending on the message content)
       - Client Version 0x0303 (TLS 1.2)
       - (Client) Random (32 bytes, generated by the TLS client’s pseudo random number generator (PRNG))
       - Number of cipher suites supported by the client (value at least 1)
       - List of cipher suites. Must include identifier for TLS\_PSK\_WITH\_AES\_128\_CBC\_SHA256 (0x00ae)
       - Extension length and Extensions (irrelevant for this example)
4. The TLS server handshake protocol responds with Server Hello and Server Hello Done messages. For the implementation employed here, each of these messages is encapsulated into a dedicated record layer frame.
5. Record layer header fields:
   * + - Content type 0x16 (Handshake)
       - Version 0x0303 (indicating TLS 1.2)
       - Length of the application data field (2 bytes, value depending on the message content)
6. Application data (“Server Hello” handshake message):
   * + - Handshake Type 0x02 (Server Hello)
       - Length of the message (3 bytes, value depending on the message content)
       - Server version 0x0303 (indicating TLS 1.2)
       - (Server) Random (32 bytes, generated by the TLS server’s PRNG)
       - Session-Id length (0x00, no session ID supplied)
       - Number of cipher suites supported by the client (at least 1)
       - Cipher suite selected by the server, shall be TLS\_PSK\_WITH\_AES\_128\_CBC\_SHA256 (0x00ae)
       - Compression method (null, no compression)
       - Extension length and Extensions (irrelevant for this example)
7. Record layer header fields:
   * + - Same as in step 2.i
8. Application data (“Server Hello Done” handshake message):
   * + - Handshake type 0x0e (Server Hello Done)
       - Length of the message (0x0000, message has no content)
9. The TLS client responds with Client Key exchange, Change Cipher Spec, Finished messages. For the implementation employed here, each of these messages is encapsulated into a dedicated record layer frame.
10. Record layer header fields:
    * + - Same as in step 2.i
11. Application data (“Client Key Exchange” handshake message):
    * + - Handshake Type 0x10 (Client Key Exchange)
        - Length of the message (3 bytes, value depending on the message content)
        - PSK client parameters:
          * Identity length ( 0x00000f in this example)
          * PSK Identity (here binary equivalent of “Client\_identity”)
12. Record layer header fields:
    * + - Content type 0x14 (Change Cipher Spec)
        - Version 0x0303 (TLS 1.2)
        - Length of the message (0x0001)
13. Application data (“Change Cipher Spec” message):
    * + - Change Cipher Spec message 0x01 (1 byte)
14. Record layer header fields:
    * + - Same as in step 2.i
15. Application data (encrypted “Finished” handshake message)
    * + - Handshake type 0x14 (Finished)
        - Length of the message 0x00000c (12)
        - Verify Data (12 bytes), see RFC 5246, section 7.4.9.
16. The server retrieves Kpsa associated with the PSK Identity, computes the master secret and authenticates the client by validating Verify Data
17. The TLS server responds with New Session Ticket, Change Cipher Spec, Finished messages. For the implementation employed here, each of these messages is encapsulated into a dedicated record layer frame.
18. Record layer header fields:
    * + - Same as in step 2.i
19. Application data (“New Session Ticket” handshake message):
    * + - Handshake Type 0x04 (New Session Ticket)
        - Length of the message (3 bytes: 0x0000b6)
        - Session Ticket:
          * Lifetime Hint ( 4 bytes: 0x00001c20, 7200 in this example)
          * Session Ticket Length (2 bytes, 0x00b0, 176 in this example)
          * Session Ticket (176 bytes), see RFC 4507, server session state enabling session resumption
20. Record layer header fields:
    * + - Content Type 0x14 (Change Cipher Spec)
        - Version 0x0303 (TLS 1.2)
        - Length of the message (0x0001)
21. Encrypted application data (“Change Cipher Spec” message):
    * + - Change Cipher Spec message 0x01 (1 byte)
22. Record layer header fields:
    * + - Same as in step 2.i
23. Application data (encrypted “Finished” handshake message, to verify that the key exchange and

authentication processes were successful):

* + - * Handshake Type 0x14 (Finished)
      * Length of the message 0x00000c (12)
      * Verify Data (12 bytes), see RFC 5246, section 7.4.9.

1. The client authenticates the server by validating Verify Data
2. Application data encrypted by the TLS record layer is exchanged between ADN-AE1 and MN-CSE



Figure A.1-1: PSK-Based Security Association Establishment

The message flow described above (excluding step 7) can be reproduced with the following commands under Linux OS using localhost IP address and port 443:

**TLS server on MN-CSE:**

$ sudo openssl s\_server -accept 443 -psk 1a2b3c4d5e6f7a8b

**TLS Client on ADN-AE1:**

$ openssl s\_client -connect 0.0.0.0:443 -psk\_identity Client\_identity \

-psk 1a2b3c4d5e6f7a8b -cipher PSK-AES128-CBC-SHA256

NOTE: The OpenSSL s\_server utility does not support table lookup of pre-shared keys when using the option

-psk\_identity AE123456789015-Lock@in.provider.com

as required for the example in clause 7.1.2. Therefore the above command line for the server includes the used PSK itself. The client command line provides the PSK identity “Client\_identity” which is expected by the server for this PSK.

Note that in order to enable Wireshark to decrypt application data which has been encrypted by the TLS record layer, it must be configured as follows:

In the Wireshark configuration menu Edit -> Preferences -> Protocols -> SSL,

1. In the “Pre-Shared-Key” field, enter Kpsa, i.e. 1a2b3c4d5e6f7a8b
2. In the (Pre)-Master-Secret log filename field, enter the name of a text file which includes Client Random (32 bytes as 64 hex characters) and the Master Secret (48 bytes as 96 hex characters) as a text line as follows:

CLIENT\_RANDOM <space> 64-characters-random <space> 96-characters-Master-Secret

The master secret is provided as log information in the terminal window, where s\_client is started. The value of Client Random can be retrieved from the Wireshark packet capture in the Client Hello handshake message.

First the data captured with Wireshark must be stored into a file. Then, after configuring Wireshark as described above, the messages in the saved data file can be decrypted by Wireshark.

*Editor’s note: relation between credential identifiers, entity identifiers and service subscription information needs to be clarified*