PPP Protocol Modules for TTCN-3 Toolset with TITAN, User Guide

Contents

[1 Introduction 2](#_Toc365900018)

[1.1 Revision history 2](#_Toc365900019)

[1.2 About this Document 2](#_Toc365900020)

[1.2.1 How to Read this Document 2](#_Toc365900021)

[1.2.2 Presumed Knowledge 2](#_Toc365900022)

[1.2.3 References 2](#_Toc365900023)

[1.2.4 Abbreviations 3](#_Toc365900024)

[1.2.5 Terminology 3](#_Toc365900025)

[1.3 System Requirements 4](#_Toc365900026)

[2 Protocol Modules 4](#_Toc365900027)

[2.1 Overview 4](#_Toc365900028)

[2.2 Installation 6](#_Toc365900029)

[2.3 Configuration 6](#_Toc365900030)

[2.3.1 Compile-time configuration 6](#_Toc365900031)

[2.3.2 Runtime Configuration 6](#_Toc365900032)

[2.4 Encoder and decoder functions 7](#_Toc365900033)

[2.5 PPP EAP functions 7](#_Toc365900034)

[2.5.1 Encoder and decoder functions 7](#_Toc365900035)

[2.5.2 Authentication and encryption key generation 8](#_Toc365900036)

[3 Example 9](#_Toc365900037)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2008-06-06 | PA1 | First draft version | ETHEKR |
| 2010-06-01 | PB1 | EAP, IP has been added | ETMEMOD |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## About this Document

### How to Read this Document

This is the User Guide for the PPP protocol module. The PPP protocol module is developed for the TTCN-3 Toolset with TITAN. This document should be read together with Product Revision Information [4] and Function Specification [5].

### Presumed Knowledge

To use this protocol module the knowledge of the TTCN-3 language [1] is essential.

### References

1. ETSI ES 201 873–1 v.3.2.1 (2007-02)  
   The Testing and Test Control Notation version 3. Part 1: Core Language
2. 2/198 17-CRL 113 200/3 Uen  
   Programmer’s Technical Reference for the TITAN TTCN-3 Test Executor
3. 1/1531-CRL 113 200/3 Uen  
   Installation Guide for the TITAN TTCN-3 Test Executor
4. 109 21-CNL 113 599–1 Uen  
   PPP Protocol Modules for TTCN-3 Toolset with TITAN, Product Revision Information
5. 155 17-CNL 113 599  
   PPP Protocol Modules for TTCN-3 Toolset with TITAN, Function Specification
6. IETF RFC 1661  
   The Point-to-Point Protocol
7. IETF RFC 1332  
   The PPP Internet Protocol Control Protocol (IPCP)
8. IETF RFC 1877  
   PPP Internet Protocol Control Protocol Extensions for   
   Name Server Addresses
9. IETF RFC 1994  
   PPP Challenge Handshake Authentication Protocol (CHAP)
10. IETF RFC 1334  
    PPP Authentication Protocols
11. IETF RFC 1662  
    PPP in HDLC-like Framing
12. IETF RFC 3748  
    Extensible Authentication Protocol (EAP)
13. Extensible Authentication Protocol Method for GSM Subscriber Identity Modules (EAP-SIM)  
    draft-haverinen-pppext-eap-sim-16.txt  
    (2004-12)
14. Extensible Authentication Protocol Method for 3rd Generation  
    Authentication and Key Agreement (EAP-AKA)  
    draft-arkko-pppext-eap-aka-15.txt  
    (2004-12)
15. IP Protocol Modules for TTCN-3 Toolset with TITAN, Function Specification

### Abbreviations

CHAP PPP Challenge Handshake Authentication Protocol

IETF Internet Engineering Task Force

IP Internet Protocol

IPCP PPP Internet Protocol Control Protocol

LCP Link Control Protocol

PAP PPP Authentication Protocols

EAP Extensible Authentication Protocol

PPP Point-to-Point Protocol

RFC Request for Comments

TTCN-3 Testing and Test Control Notation version 3

### Terminology

TITAN TTCN-3 Test Executor

## System Requirements

Protocol modules are a set of TTCN-3 source code files that can be used as part of TTCN-3 test suites only. Hence, protocol modules alone do not put specific requirements on the system used. However in order to compile and execute a TTCN-3 test suite using the set of protocol modules the following system requirements must be satisfied:

* TITAN TTCN-3 Test Executor installed. For installation guide see [3].

# Protocol Modules

## Overview

Protocol modules implement the message structures of the corresponding protocol in a formalized way, using the standard specification language TTCN-3. This allows defining of test data (templates) in the TTCN-3 language [1] and correctly encoding/decoding messages when executing test suites using the TITAN TTCN-3 test environment.

Protocol modules are using TITAN’s RAW encoding attributes [2] and hence are usable with the TITAN test toolset only.

The PPP messages are represented by the TTCN-3 record type PDU\_PPP.

The information field of PDU\_PPP is a TTCN-3 union which contains the appropriate associated subtype IP, LCP, IPCP, CHAP, PAP or EAP.

The implemented IP messages in TTCN-3 are defined in [15].

The implemented LCP messages in TTCN-3 are:

|  |
| --- |
| LCP\_ConfigureRequest |
| LCP\_ConfigureAck |
| LCP\_ConfigureNak |
| LCP\_ConfigureReject |
| LCP\_TerminateRequest |
| LCP\_TerminateAck |
| LCP\_CodeReject |
| LCP\_ProtocolReject |
| LCP\_EchoRequest |
| LCP\_EchoReply |
| LCP\_DiscardRequest |

The implemented IPCP messages in TTCN-3 are:

|  |
| --- |
| IPCP\_ConfigureRequest |
| IPCP\_ConfigureAck |
| IPCP\_ConfigureNak |
| IPCP\_ConfigureReject |
| IPCP\_TerminateRequest |
| IPCP\_TerminateAck |
| IPCP\_CodeReject |

The implemented CHAP messages in TTCN-3 are:

|  |
| --- |
| CHAP\_Challenge |
| CHAP\_Response |
| CHAP\_Success |
| CHAP\_Failure |

The implemented PAP messages in TTCN-3 are:

|  |
| --- |
| PAP\_AuthenticateRequest |
| PAP\_AuthenticateAck |
| PAP\_AuthenticateNak |

Using these type records, templates can be defined to send and receive a given message.

The EAP\_Types.ttcn module contains the implemented PPP EAP messages. These messages have the same stucture as these have in the RADIUS Test Port:

|  |
| --- |
| eap\_identity |
| eap\_notification |
| eap\_nak |
| eap\_md5\_challenge |
| eap\_one\_time\_password |
| eap\_generic\_token\_card |
| eap\_sim |
| eap\_aka |

## Installation

The set of protocol modules can be used in developing TTCN-3 test suites using any text editor. However to make the work more efficient a TTCN-3-enabled text editor is recommended (e.g. nedit, xemacs). Since the PPP protocol module is used as a part of a TTCN-3 test suite, this requires TTCN-3 Test Executor and a C compiler be installed before the module can be compiled and executed together with other parts of the test suite. For more details on the installation of TTCN-3 Test Executor see the relevant parts of [2]

## Configuration

### Compile-time configuration

The compile-time configuration of the RADIUS test port is performed by customizing the generated Makefile. The following steps must be made:

1. The OPENSSL\_DIR variable must be set to the location of OpenSSL.

2. The CPPFLAGS must contain –I$(OPENSSL\_DIR)/include

3. The operation specific libraries (e. g. SOLARIS\_LIBS) should contain –lresolv.

### Runtime Configuration

The behaviour of the executable test program is determined by the run-time configuration file. This is a simple text file, which contains various sections (e.g. [TESTPORT\_PARAMETERS]) after each other. The usual suffix of configuration files is .cfg. Only the [MODULE\_PARAMETERS] section is related to the PPP (EAP) protocol module. In this section you can specify parameters that are passed to the protocol module.

The following parameters are allowed:

**tsp\_skip\_auth\_enc**r (OPTIONAL): If this parameter is set to “true”, the authentication and encryption functionality of EAP-SIM and EAP-AKA is turned off.

**tsp\_global\_keying** (OPTIONAL): If it is set to “false”, the protocol module uses EAP-Identifier based keying material for EAP-SIM and EAP-AKA, ie., keying material is treated separately for each EAP-Identifier. If this parameter is set to “true”, then the test port uses global keying with EAP-SIM and EAP-AKA (a pseudo-value 256 is used as EAP-Identifier). The default value is “false”.

**tsp\_debugging** (OPTIONAL): This boolean parameter allows the output of textual debug information of TTCN-3 “log” statements on the console or in log file (depending on the setting of consoleMask and fileMask parameters). Note that error messages for serious errors are not affected by the tsp\_debugging parameter. The default value is “false”.

**tsp\_SIM\_Ki** (OPTIONAL): The length of this octetstring parameter is 16 octet. Ki key has to be set prior to sending or receiving EAP-SIM messages containing AT\_ENCR\_DATA. The Ki key will be set automatically with tsp\_SIM\_Ki if it is not set with f\_set\_Ki function.  
The default value is '00112233445566778899AABBCCDDEEFF'O

**tsp\_AKA\_K** (OPTIONAL): The length of this octetstring parameter is 16 octet. K key has to be set prior to sending or receiving EAP-AKA messages containing AT\_ENCR\_DATA. The AKA K key will be set automatically with tsp\_AKA\_K if it is not set with f\_set\_K function.  
The default value is '0123456789ABCDEF0123456789ABCDEF'O

**tsp\_AKA\_SQN** (OPTIONAL): The length of this octetstring parameter is 6 octet. SQN key has to be set prior to sending EAP-AKA messages containing AT\_AUTN. The AKA SQN key will be set automatically with tsp\_AKA\_SQN if it is not set with f\_set\_SQN function.  
The default value is '000000000000'O

**tsp\_AKA\_SQN\_MS** (OPTIONAL): The length of this octetstring parameter is 6 octet. SQN\_MS key has to be set prior to sending EAP-AKA messages containing AT\_AUTS. The AKA SQN\_MS key will be set automatically with tsp\_AKA\_SQN\_MS if it is not set with f\_set\_SQN\_MS function.  
The default value is '000000000000'O

**tsp\_AKA\_AMF** (OPTIONAL): The length of this octetstring parameter is 2 octet. AMF key has to be set prior to sending EAP-AKA messages containing AT\_AUNT. The AKA AMF key will be set automatically with tsp\_AKA\_AMF if it is not set with f\_set\_AMF function.  
The default value is '0000'O

## Encoder and decoder functions

The following encoder/decoder functions are available which provide for the correct encoding of messages when sent from TITAN and correct decoding of messages when received by TITAN.:

Name Type of formal parameters Type of return value  
**enc\_PDU\_PPP PDU\_PPP octetstring  
dec\_PDU\_PPP octetstring PDU\_PPP**

Note that the Address and Control fields defined in [11] are treated as a single optional field at the beginning of PDU\_PPP.

## PPP EAP functions

### Encoder and decoder functions

Name Type of parameters Type of return value

**f\_enc\_PDU\_EAP PDU\_EAP octetstring**

**f\_dec\_PDU\_EAP octetstring PDU\_EAP**

**f\_enc\_PDU\_EAP\_list PDU\_EAP\_list octetstring**

**f\_dec\_PDU\_EAP\_list octetstring PDU\_EAP\_list**

**f\_enc\_eap\_sim\_attrib\_list eap\_sim\_attrib\_list octetstring**

**f\_dec\_eap\_sim\_attrib\_list octetstring eap\_sim\_attrib\_list**

**f\_enc\_eap\_aka\_attrib\_list eap\_aka\_attrib\_list octetstring**

**f\_dec\_eap\_aka\_attrib\_list octetstring eap\_aka\_attrib\_list**

### Authentication and encryption key generation

EAP\_port\_descriptor stores the authentication and encryption keys. It is needed to be initialized; during the use of a descriptor variable without initialization can occur errors!

function f\_initEAPPortDescriptor(inout EAP\_port\_descriptor descriptor);

Function for automatic generation and storage of authentication and encryption keys:

function f\_get\_EAP\_parameters(inout octetstring pl\_ext\_eap\_message,inout EAP\_port\_descriptor pl\_descriptor,in boolean incoming\_message)

Function for generating AT\_MAC, Kaut key is needed:

function f\_calc\_HMAC(in octetstring key, in octetstring input, in integer out\_length) return octetstring;

The following functions set the keys for *identifier:*

function f\_set\_Ki(in integer identifier, in octetstring input, inout EAP\_port\_descriptor descriptor);

function f\_set\_K(in integer identifier, in octetstring input, inout EAP\_port\_descriptor descriptor);

function f\_set\_SQN(in integer identifier, in octetstring input, inout EAP\_port\_descriptor descriptor);

function f\_set\_SQN\_MS(in integer identifier, in octetstring input, inout EAP\_port\_descriptor descriptor);

function f\_set\_AMF(in integer identifier, in octetstring input, inout EAP\_port\_descriptor descriptor);

The function below calculates XDOUT, Kencr, Kaut and AK values. Kaut is used when calculating MAC values, Kencr is used for encryption and decryption of AT\_ENCR\_DATA attributes, and AK is used for calculating and verifying AT\_AUTN and AT\_AUTS values.

function f\_calc\_AKA\_Keys(in octetstring pl\_eap\_identity, in octetstring pl\_AKA\_K,in octetstring pl\_rand, inout octetstring pl\_AK,inout octetstring pl\_Kaut,inout octetstring pl\_Kencr) return octetstring

A3A8 value is generated from Ki key and rand list. It is used in calculating Kaut:

function f\_calc\_A3A8(in octetstring key,in octetstring rand)return octetstring;

The value n\*SRES is n SRES values concatenated. It can be generated with the following function from Ki key and rand list:

function f\_calc\_SRES(in octetstring key,in octetstring rand)return octetstring;

When generating Kaut and Kenc the input octetstring is concatenated from identifier, A3A8, nonce\_mt, version list and selected version.

function f\_calc\_Kaut(in octetstring input,inout octetstring kencr) return octetstring;

The next function is used in f\_crypt\_atSimEncrData and f\_crypt\_atAKAEncrData. It generates AES\_cbc\_encrypted or decrypted value. Kenc key and ivec is needed for calculation.

function f\_encrypt\_at\_encr(in octetstring key,in octetstring input,in octetstring ivec,in boolean decrypt) return octetstring;

Functions for encryption or decryption. Kenc and ivec is needed.

function f\_crypt\_atSimEncrData( in at\_sim\_encr\_data pl\_encr\_data, in octetstring key,in octetstring ivec,in boolean decrypt) return at\_sim\_encr\_data;

function f\_crypt\_atAKAEncrData(in at\_aka\_encr\_data pl\_encr\_data, in octetstring key,in octetstring ivec,in boolean decrypt)return at\_aka\_encr\_data;

# Example

There are no examples available for this protocol module.