|  |  |
| --- | --- |
|  |  |

RADIUS Protocol Module Generator for TTCN-3 Toolset with TITAN, Description

Contents

[1 About this Document 2](#_Toc393370936)

[1.1.1 How to Read this Document 2](#_Toc393370937)

[1.1.2 Presumed Knowledge 2](#_Toc393370938)

[2 Functionality 3](#_Toc393370939)

[2.1 Product contents, structure 3](#_Toc393370940)

[2.2 Protocol version implemented 3](#_Toc393370941)

[2.3 Routing Functionality 4](#_Toc393370942)

[2.4 Modified and non-implemented Protocol Elements 4](#_Toc393370943)

[2.5 Ericsson-specific changes 4](#_Toc393370944)

[2.6 Backward incompatibilities 4](#_Toc393370945)

[2.7 System Requirements 5](#_Toc393370946)

[3 Feature list 5](#_Toc393370947)

[3.1 Background 5](#_Toc393370948)

[3.2 Architecture 6](#_Toc393370949)

[3.3 Naming Conventions 7](#_Toc393370950)

[3.3.1 Key to unique naming of identifiers 8](#_Toc393370951)

[3.3.2 How to model enumeration type Attributes in RDFs? 8](#_Toc393370952)

[3.3.3 How to model code enumeration type in RDFs? 9](#_Toc393370953)

[3.4 Vendor Specific attribute handling 9](#_Toc393370954)

[3.4.1 Vendor ID definition 9](#_Toc393370955)

[3.4.2 Mandatory type definitions 9](#_Toc393370956)

[3.5 Script operation 10](#_Toc393370957)

[3.5.1 Load and parse all input files 10](#_Toc393370958)

[3.5.2 Type identifiers 10](#_Toc393370959)

[3.5.3 Attrib 11](#_Toc393370960)

[3.5.4 Attrib\_Data 11](#_Toc393370961)

[3.5.5 GenericAttrib 11](#_Toc393370962)

[3.5.6 GenAttrib 12](#_Toc393370963)

[3.5.7 Attribs 12](#_Toc393370964)

[3.5.8 Code (Packet type) 12](#_Toc393370965)

[3.5.9 Output TTCN-3 module 13](#_Toc393370966)

[3.6 Using the scipt 13](#_Toc393370967)

[4 Functional specification 13](#_Toc393370968)

[4.1 Product contents, structure 13](#_Toc393370969)

[4.2 Encoding/Decoding and Other Related Functions 13](#_Toc393370970)

[4.3 Protocol version implemented 14](#_Toc393370971)

[5 Protocol Modules 15](#_Toc393370972)

[5.1 Overview 15](#_Toc393370973)

[5.2 Installation 16](#_Toc393370974)

[5.3 Compilation 16](#_Toc393370975)

[5.4 Helper functions 18](#_Toc393370976)

[6 Upgrading templates used by the RADIUS test port 20](#_Toc393370977)

[7 Error handling 21](#_Toc393370978)

[8 Examples 22](#_Toc393370979)

[8.1 Mapping module 22](#_Toc393370980)

1. [8.1.1.1 Configuration 23](#_Toc393370981)

[8.1.2 Client mode 24](#_Toc393370982)

1. [8.1.2.1 Overview 24](#_Toc393370983)

[8.1.3 Server mode 25](#_Toc393370984)

1. [8.1.3.1 Configuration 25](#_Toc393370985)

[8.1.4 Attributes of the RADIUSmsg\_PT port 25](#_Toc393370986)

[8.2 Demo module 26](#_Toc393370987)

[8.2.1 Test cases 26](#_Toc393370988)

[8.2.2 Configuration files 27](#_Toc393370989)

[8.2.3 Examples for building the project 27](#_Toc393370990)

[8.2.4 Script to modify Makefile 27](#_Toc393370991)

[9 Terminology 28](#_Toc393370992)

[9.1 Abbreviations 28](#_Toc393370993)

[10 References 29](#_Toc393370994)

[11 Change information 30](#_Toc393370995)

[11.1 R6A 30](#_Toc393370996)

[11.2 R9A 30](#_Toc393370997)

[11.3 R10A 30](#_Toc393370998)

[11.4 R11A 30](#_Toc393370999)

# About this Document

### How to Read this Document

This is the Function Specification of the RADIUS Protocol Module Generator. The RADIUS Protocol Module Generator is developed for the TTCN-3 Toolset with TITAN. This document should be read together with Product Revision Information [3].

### Presumed Knowledge

To use this protocol module the knowledge of the TTCN-3 language [2] is essential.

Basic knowledge of the RADIUS protocol [1] is valuable to use this protocol module.

# Functionality

## Product contents, structure

The major parts of RPMG are:

1. ATTR.awk script: This is the most important part of the product.
2. A pair of encoder and decoder functions to invoke RAW encoder/decoder.
3. RADIUS\_Mapping.ttcn modul: provide the authentication.

The RADIUS Base Protocol [1] and other RADIUS applications are specified in RDFs developed by TCC as part of the RPMG product.

The TTCN-3 module (RADIUS\_Types) that is generated by the script varies between applications, thus it is NOT a product.

## Protocol version implemented

Currently the following applications are supported:

|  |  |
| --- | --- |
| **RDFs** | **Refs.** |
| BaseTypes\_IETF\_RFC2865.rdf  Base\_IETF\_RFC2865.rdf | [1] |
| Accounting\_IETF\_RFC2866\_RFC2867.rdf | [7][14] |
| TunnelAuthentication\_IETF\_RFC2868.rdf | [8] |
| Extensions\_IETF\_RFC2869.rdf | [9] |
| IPv6\_IETF\_RFC3162.rdf | [10] |
| DynamicAuthorizationExtensions\_IETF\_RFC5176.rdf | [11] |
| ChargeableUserIdentity\_IETF\_RFC4372.rdf | [12] |
| GGSN\_FS\_46\_15517\_CSA113\_4RevB.rdf | [13] |

|  |  |
| --- | --- |
| **Vendor specific RDFs** | **Refs.** |
| Cisco.rdf |  |
| Juniper.rdf |  |
| Scap.rdf |  |
| Smartedge.rdf | [15] |
| Vendor.rdf |  |
| skt.rdf |  |
| Vendor\_detailed\_ericsson.rdf |  |

|  |  |
| --- | --- |
| **RDFs (in obsolate)** | **Refs.** |
| RadiusBaseTypes.rdf  RadiusBaseAttributes.rdf | [1] |
| RadiusAccountingAttributes.rdf | [7] |
| TunnelAuthenticationAttributes.rdf | [8] |
| RadiusExtensions.rdf | [9] |
| RadiusAndIPv6.rdf | [10] |
| RadiusAuthExtensions.rdf | [11] |
| RadiusGGSN.rdf | [13] |

## Routing Functionality

Routing functionality is not performed.

## Modified and non-implemented Protocol Elements

-

## Ericsson-specific changes

There is no Ericsson specific change in this product.

## Backward incompatibilities

-

## System Requirements

The RADIUS protocol module generator consist of several RDF files, contains different application definitions of RADIUS protocols and an AWK script which read the RDF files and generates the RADIUS\_Types.ttcn type definition module.

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 R7A (1.7.pl0) or higher installed. For installation guide see [16]. Please note: This version of the protocol module is not compatible with TITAN releases earlier than R7A.
* The RADIUS protocol module generator contains functions, use MD5 checksum, so the same OpenSSL must be installed as used in TITAN. For installation guide see [18].

# Feature list

## Background

Former solution for testing RADIUS applications is based on the RADIUS Test Port [5]. This test port implies a number of limitations:

1. Attributes and other application-specific data are hard-coded in the Test Port, which makes the extension hard. New Attributes need to be added, encoded and decoded manually.
2. The support of different revisions of same RADIUS application may required by different projects. Different revisions may contain e.g. overlapping Attribute codes or other contradictory type definitions, which can only be handled using run-time switches.

RPMG provides solution to this problem by dynamically generating the type definition module containing the Attributes and definitions of the chosen applications.

The generated protocol module implements the message structures of the RADIUS protocol in a formalized way, using the standard specification language TTCN-3. This allows defining of test data (templates) in the TTCN-3 language [2] and correctly encoding/decoding messages when executing test suites using the Titan TTCN-3 test environment.

The protocol module uses Titan’s RAW encoding attributes [4] for RADIUS encoder and hence is usable with the Titan test toolset only.

## Architecture

The dynamically generated protocol module is assembled from one or more RDFs using a script (see Figure 1). The RDFs – which are not valid TTCN-3 modules – contain TTCN-3 type definitions of Attributes of RADIUS applications. One RDF MUST NOT contain more than one application but applications MAY be split into several RDFs.



Figure 1 RPMG architecture

The naming of RDFs should follow the <Official-Application-Name>Attributes\_ <Application-Version> scheme. The extension MUST NOT be .ttcn; .rdf is recommended.

If the application uses only a minor sub-set of some standard specification then it is acceptable to define these Attributes in the framework of the new application instead of including all unnecessary Attributes of the standard.

The type definitions for testing protocols complying the RADIUS Base Specification are split in two RDFs:

|  |  |
| --- | --- |
| **RDF name** | **Contents** |
| RadiusBaseTypes | Remote Authentication Dial In User Service [1] related type definitions |
| RadiusBaseAttributes | Remote Authentication Dial In User Service [1] Attributes type definitions |

Or using the latest RDF interface:

|  |  |
| --- | --- |
| **RDF name** | **Contents** |
| BaseTypes\_IETF\_RFC2865.rdf | Remote Authentication Dial In User Service [1] related type definitions |
| Base\_IETF\_RFC2865.rdf | Remote Authentication Dial In User Service [1] Attributes type definitions |

The ATTR.awk script merges its argument RDFs into a single valid TTCN-3 module called RADIUS\_Types by default.

The input RDFs must comply with the naming and typographical conventions described herein in order for the ATTR.sh script to produce a syntactically and semantically valid TTCN3 module.

The script can be optionally invoked with an option. This option modifies the script operation. The option must appear in the argument list before the RDF files. It is introduced with the –v flag and must not immediately follow the –v flag; whitespace separation is required. The option itself must be written in one word.

The supported option is:

old\_structured\_code: This option results the original structured RADIUS\_Types generated module. This option is disabled by default.

module\_id: This option can be used to alter the identifier of generated TTCN-3 module. The module\_id shall be a valid TTCN-3 identifier.

use\_application\_revision: This option results the application revision string to be added as prefix to generated identifiers. The application revision string is hardcoded into the input DDF files. This option is disabled by default.

enum\_2\_UnsignedInt: This option makes the script to replace all enumeration type attributes, which are found in input RDF files, to typeUnsigned32. This option is disabled by default.

## Naming Conventions

The generated identifiers of types are based on information provided in comments within the RDFs themselves.

1. Applications are distinguished using the unique <Application-Name>, which is assigned by TCC. The <Application-Name> is used to prefix type as well as certain field identifiers in the generated RADIUS\_Types module to ensure unique naming. The <Application Revision> is only optionally used in prefixes. The <Application-Name> and <Application Revision> are hard-coded in each RDF file using the following format:

// APPLICATION-NAME: Auth

// APPLICATION-REVISION: RFC2868

1. Attribute properties (e.g. name, code) are enlisted in C++ style comment right before the Attribute type definition using the following format:

// Attrib: <Official-Attribute-Name> (<Official-Attribute-Code>)

type Type\_Specifier Official\_Attribute\_Name …

1. The entire comment line with the information MUST stand in the same line (no line breaks when it splits to multiple lines).
2. The TTCN-3 type definition following a properly formed comment line is interpreted as an Attribute definition, if the type identifier matches the <Official-Attribute-Code> appearing in the comment before.
3. The <Official-Attribute-Name> and <Official-Attribute-Code> shall come from the relevant RFC, IETF Draft or other specification.
4. The TTCN-3 identifiers used in <Official-Attribute-Name> must keep the original naming except when this collides with TTCN-3 identifier’s naming rules:
5. Hyphens and spaces must be replaced by a single underscore
6. Trailing "Attribute" SHOULD be omitted if not part of the name
7. The <Official-Attribute-Code> must be given as integer numbers!

### Key to unique naming of identifiers

The following uniqueness criteria – derived from RADIUS [1] – must hold for identifiers used in RDFs:

1. <Application-Name> MUST be globally unique: each application identifier must be formulated so that it is always unique.   
     
   Example <Application-Name>s: BASE, Acc
2. <Official-Attribute-Name> may not be unique: It happens that the same Attribute name is used in the same or in different RADIUS applications. The script is designed to cope with this, thus it is recommended to keep the standard Attribute name with respect to naming conventions.
3. <Official-Attribute-Code> MUST be globally unique (except within different revisions of the same application!) since this 8Bits number determines the Attribute.

### How to model enumeration type Attributes in RDFs?

It is important to ensure the unique naming of enumeration type identifiers and enumeration items. The enumerated type Attributes require one type definition: The enumerated type definition containing the valid enumeration items. The identifier of the enumeration type shall be <Official-Attribute-Name>.

All enumerations in RDFs get the following attributes automatically assigned to enumeration type Attributes' enumerations:

with {

variant "FIELDLENGTH(X)"

variant "BYTEORDER(last)"

variant "COMP(2scompl)"

}

The FIELDLENGTH attribute depends on the length of the field. The enumeration type identifiers and enumeration items are variable in length; these must be defined in RDFs. The type name contains the length in the following way: the enumerated word is replaced with the enum, and the length of the field is written after the enum, and there must be an underline char between the enum word and the length of the enumeration type.

The following example shows how to define for example a 8-bit-long enumerated type Attribute:

// Attrib: <Official-Attribute-Name> (<Official-Attribute-Code>)

type **enum\_8** Official\_Attribute\_Name

{

…

}

A 32-bit-long enumeration type can be defined otherwise, simply use the word: enumerated.

### How to model code enumeration type in RDFs?

Code enumeration type can be extended in RADIUS applications. RPMG merges them together into a single type definition with proper attributes.  
The generated code enumeration type is based on information provided in comments within the RDFs themselves.

Code properties (e.g. name, code) are enlisted in C++ style comment using the following format:

// Packet-Type: <Official-Code-Name> (<Official-Code-Code>)

## Vendor Specific attribute handling

The vendor id and the details of the type definition of the vendor specific attributes can be defined in the RDFs.

### Vendor ID definition

The vendor id and the vendor name should be defined in the following forms:

// Vendor: <Vendor\_name> (<Vendor-Code>)

Example:

// Vendor: ericsson (193)

### Mandatory type definitions

The following types should be defined for a vendor specific attributes:

type set of <Vendor\_name>\_type <Vendor\_name>\_subattr\_list;

type enum\_8 <Vendor\_name>\_type\_enum{

...

}

type record <Vendor\_name>\_type

{

<Vendor\_name>\_type\_enum f\_<Vendor\_name>\_type,

UINT8 attrib\_length\_spec,

<Vendor\_spec\_type> string\_val

} with {

variant (attrib\_length\_spec) "LENGTHTO(f\_<Vendor\_name>\_type, attrib\_length\_spec, string\_val)"

}

The <Vendor\_spec\_type> can be:

* octetsring
* vendor\_specific\_value, defined in BaseTypes\_IETF\_RFC2865.rdf
* other user specified type

## Script operation

The TTCN-3 module, containing all relevant type definitions, is generated automatically from the relevant RDFs by a script. This will ensure that no collision can appear between proper RADIUS applications.

The top-level RADIUS PDU to send/receive is always PDU\_RADIUS.

### Load and parse all input files

If overlapping Attribute codes (same Attribute code) are found during parsing of RDFs then the created RADIUS\_Types module will not compile!

In this case at the end of the RADIUS\_Types module the following error message has been appeared:

ERROR: attrib\_descriptors () != matching\_attrib\_types ()

with the number of elements in the brackets.

### Type identifiers

The script changes Attribute type identifiers in order to avoid name collisions. The <Official-Application-Name> will prefix the Official\_Attribute\_Name defined in RDF. Example:

Attribute type definition in RDF:

// RFC 2865

// Attrib: User-Name (1)

type Attrib\_String User\_Name;

The corresponding type definition in the generated module:

// RFC 2865

// Attrib: User-Name (1)

type Attrib\_String Base\_User\_Name

In the original styled RADIUS\_Types module (using old\_structured\_code option of the awk script) there are generated Attribute types. These types are not included in new styled RADIUS\_Types module. Each generated Attribute type is a record consists of three fields: the type attrib\_type, the length attrib\_length and the field with the value of Attribute.

Example:

type record Attrib\_Base\_User\_Name

{

Attrib attrib\_type,

UINT8 attrib\_length,

Base\_User\_Name base\_user\_name

} with {

variant "PRESENCE (attrib\_type=Base\_User\_Name)"

variant (attrib\_length) "LENGTHTO(attrib\_type, attrib\_length, base\_user\_name)"

}

### Attrib

All Attributes code is in a single enumerated type. Attrib enumeration type is merged together from the Attributes definitions of the RDF files by the AWK script. All enumeration item defined in different application are collected together and written to the generated RADIUS\_Types.ttcn file. Proper encoding attributes are added to the Attrib type by the script.

type enumerated Attrib

{

Base\_User\_Name (1),

…

} with {

variant "FIELDLENGTH(8)"

variant "BYTEORDER(last)"

variant "COMP(2scompl)"

}

### Attrib\_Data

There is Attrib\_Data type in the new styled RADIUS\_Types module. This type is not included in original styled RADIUS\_Types module. The Attrib\_Data type is a generated union type containing all Attribute types found in the RDFs:

type union Attrib\_Data

{

Base\_User\_Name base\_user\_name,

…,

octetstring attrib\_UNKNOWN

}

### GenericAttrib

In the original styled RADIUS\_Types module (using old\_structured\_code option of the awk script) the GenericAttrib type is a generated union type containing all Attribute types found in the RDFs:

type union GenericAttrib {

Attrib\_<Application-Name>\_<Official-Attribute-Name>

attrib\_<Application-Name>\_<Official-Attribute-Name>,

Attrib\_UNKNOWN attrib\_UNKNOWN

}

attrib\_UNKNOWN will contain the erroneous Attribute when something went wrong during the decoding of the Attribute data (in the original and the new style generated code).

type record Attrib\_UNKNOWN

{

UINT8 attrib\_type,

UINT8 attrib\_length,

octetstring attrib\_value

} with {

variant (attrib\_length) "LENGTHTO(attrib\_type, attrib\_length, attrib\_value)"

}

In the new styled RADIUS\_Types module the GenericAttrib type is a record consists of three fields: the type attrib\_type, the length attrib\_length and the Attrib\_Data typed attrib\_data.

type record GenericAttrib

{

Attrib attrib\_type,

UINT8 attrib\_length,

Attrib\_Data attrib\_data

} with {

variant (attrib\_length) "LENGTHTO(attrib\_type, attrib\_length, attrib\_data)"

variant (attrib\_data) "CROSSTAG(

base\_user\_name,attrib\_type=Base\_User\_Name;

…;

attrib\_UNKNOWN, OTHERWISE

)"

}

### GenAttrib

There is GenAttrib type in the new styled RADIUS\_Types module. This type is not included in original styled RADIUS\_Types module. The GenAttrib type is a generated union type containing the generic\_Attrib and the attrib\_UNKNOWN.

type union GenAttrib

{

GenericAttrib genericAttrib,

Attrib\_UNKNOWN attrib\_UNKNOWN

}

### Attribs

Attribs is a list of Attributes (set of GenericAttrib) in the original styled RADIUS\_Types module.

Attribs is a list of GenAttrib in the new styled RADIUS\_Types module.

### Code (Packet type)

Code enumeration type is merged together from the RDF file of different application’s Code definition by the AWK script. All enumeration item defined in different application are collected together and written to the generated RADIUS\_Types.ttcn file. Proper encoding attributes are added to the Code type by the script.

### Output TTCN-3 module

All definitions of RDF files, which are not subject to change are written to the output RADIUS\_Types.ttcn file as is.

## Using the scipt

The ATTR.awk script can be executed with GNU AWK [6] and its output MUST be redirected into "RADIUS\_Types.ttcn"!

# Functional specification

## Product contents, structure

The major parts of RPMG are:

1. ATTR.awk script: This is the most important part of the product.
2. A pair of encoder and decoder functions to invoke RAW encoder/decoder.
3. RADIUS\_Mapping.ttcn modul: provide the authentication.

The RADIUS Base Protocol [1] and other RADIUS applications are specified in RDFs developed by TCC as part of the RPMG product.

The TTCN-3 module (RADIUS\_Types) that is generated by the script varies between applications, thus it is NOT a product.

## Encoding/Decoding and Other Related Functions

This product also contains encoding/decoding functions that assure correct encoding of messages when sent from Titan and correct decoding of messages when received by Titan. Implemented encoding/decoding functions:

Name Type of formal parameters Type of return value  
f\_RADIUS\_Enc (in PDU\_RADIUS pdu) octetstring;

f\_RADIUS\_Dec (in octetstring stream) PDU\_RADIUS;

Two separate external functions are available for encrypting the password and calculating the Authenticator:

Name Type of formal parameters Type of return value  
f\_crypt\_password (in octetstring P, octetstring;  
 in octetstring req\_auth,  
 in octetstring salt,  
 in boolean decrypt,  
 in charstring secret)

f\_calc\_MD5 (in octetstring input) octetstring;

f\_crypt\_s\_key (in octetstring pl\_s\_key, octetstring  
 in octetstring pl\_req\_auth  
 in charstring secret  
 in boolean decrypt)

f\_crypt\_tunnel\_password (in octetstring pl\_password octetstring  
 in octetstring req\_auth  
 in octetstring salt  
 in charstring secret  
 in Boolean decrypt)

Other functions are available in PPP\_CNL113599 Protocol Module to encode and decode EAP Messages which can be encapsulated in EAP-Message Attribute (Type: 79)

## Protocol version implemented

Currently the following applications are supported:

|  |  |
| --- | --- |
| **RDFs** | **Refs.** |
| BaseTypes\_IETF\_RFC2865.rdf  Base\_IETF\_RFC2865.rdf | [1] |
| Accounting\_IETF\_RFC2866\_RFC2867.rdf | [7][14] |
| TunnelAuthentication\_IETF\_RFC2868.rdf | [8] |
| Extensions\_IETF\_RFC2869.rdf | [9] |
| IPv6\_IETF\_RFC3162.rdf | [10] |
| DynamicAuthorizationExtensions\_IETF\_RFC5176.rdf | [11] |
| ChargeableUserIdentity\_IETF\_RFC4372.rdf | [12] |
| GGSN\_FS\_46\_15517\_CSA113\_4RevB.rdf | [13] |

|  |  |
| --- | --- |
| **Vendor specific RDFs** | **Refs.** |
| Cisco.rdf |  |
| Juniper.rdf |  |
| Scap.rdf |  |
| Smartedge.rdf | [15] |
| Vendor.rdf |  |
| skt.rdf |  |
| Vendor\_detailed\_ericsson.rdf |  |

|  |  |
| --- | --- |
| **RDFs (in obsolate)** | **Refs.** |
| RadiusBaseTypes.rdf  RadiusBaseAttributes.rdf | [1] |
| RadiusAccountingAttributes.rdf | [7] |
| TunnelAuthenticationAttributes.rdf | [8] |
| RadiusExtensions.rdf | [9] |
| RadiusAndIPv6.rdf | [10] |
| RadiusAuthExtensions.rdf | [11] |
| RadiusGGSN.rdf | [13] |

# Protocol Modules

## Overview

Protocol modules implement the message structure of the related protocol in a formalized way, using the standard specification language TTCN-3. This allows defining of test data (templates) in the TTCN-3 language [4] and correctly encoding/decoding messages when executing test suites using the Titan TTCN-3 test environment.



Figure 1 RPMG Architecture

The RADIUS protocol module (RADIUS\_Types.ttcn) is generated dynamically from the needed RDF files with the help of an AWK script (Figure 1). The produced protocol module uses Titan’s RAW encoding attributes [3] for encoding and decoding and hence it is usable with the Titan test toolset only.

## Installation

The set of protocol modules can be used for 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 RADIUS protocol is used as a part of a TTCN-3 test suite, this requires Titan TTCN-3 Test Executor 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 section of [16].

The ATTR.sh shell script runs on Bourne Shell, which is usually available on all UNIX like workstations. The ATTR.awk script, which processes the RDF files and creates the RADIUS protocol module, can be executed with GNU AWK version 3.1.6 or later [6] so it must be available on the system.

## Compilation

First you need to obtain the required RDF files. After you have the RDF files containing the definitions of the selected RADIUS applications, you can generate the proper RADIUS\_Types.ttcn module by issuing the following command:

ATTR.sh RadiusBaseAttributes.rdf RadiuaBaseTypes.rdf *OtherApplications.rdf*

or

gawk -f ATTR.awk RadiusBaseAttributes.rdf RadiuaBaseTypes.rdf *OtherApplications.rdf* > RADIUS\_Types.ttcn

The script can be optionally invoked with some options. This options modify the script operation. The options must appear in the argument list before the RDF files. It is introduced with the –v flag and must not immediately follow the –v flag; whitespace separation is required. The option itself must be written in one word.

The supported option is:

old\_structured\_code: This option results the original structured RADIUS\_Types generated module. This option is disabled by default.

module\_id: This option can be used to alter the identifier of generated TTCN-3 module. The module\_id shall be a valid TTCN-3 identifier.

use\_application\_revision: This option results the application revision string to be added as prefix to generated identifiers. The application revision string is hardcoded into the input DDF files. This option is disabled by default.

enum\_2\_UnsignedInt: This option makes the script to replace all enumeration type attributes, which are found in input RDF files, to typeUnsigned32. This option is disabled by default.

The next command stores the generated TTCN-3 definitions in original structured module XYZ and translates all enumeration type attributes to a proper unsigned integer.

ATTR.sh –v old\_structured\_code=1 –v module\_id=XYZ –v enum\_2\_UnsignedInt RadiusBaseAttributes.rdf RadiuaBaseTypes.rdf *OtherApplications.rdf*

In case you want to add the task of generation of RADIUS\_Types.ttcn module into your Makefile, you should do the following:

1. Generate the GNU Makefile for your existing TTCN-3 and C++ files, except RADIUS\_Types.ttcn.
2. Add RADIUS\_Types.ttcn manually to the list of the TTCN-3 sources into the generated Makefile.
3. Add the following rules to your Makefile:  
     
   AWK=/usr/bin/gawk  
     
   RADIUS\_Types.ttcn: RadiusBaseTypes.rdf RadiusSBaseAttributes.rdf <Input FILEs containing Attribute definitions> ATTR.awk  
    $(AWK) -f ATTR.awk RadiusBaseTypes.rdf RadiusSBaseAttributes.rdf <Input FILEs containing Attribute definitions> > $@  
     
   The location of the AWK depends on the operation system and the distribution.

When you use GUI for building executable, on top of normal project creation you should take care of the following:

1. Add ATTR.awk script and all RDF files you need to add to the misc files section. Please note that you should generate softlinks to the build directory with selecting the files and select ‘Generate Softlinks’ menu item manually as it is not generated automatically by the GUI.
2. Create a script to modify the generated Makefile (see clause 5.2.4 for example). List all the RDF files you need when executing the AWK script.
3. Add script to modify the Makefile in the project properties.
4. If you want to have the RADIUS\_Types.ttcn file added to your project, you can, but after adding, you should exclude the file from build in order to avoid double occurrence in the Makefile.

## Helper functions

The following external functions are available to calculate the Request Authenticator field and check the Response Authenticator field; encrypt and decrypt the password attributes:

external function f\_calc\_MD5(in octetstring input) return octetstring;

This function calculates the MD5 hash.

It is important to mention, that if the Authenticator field is set to 0, the encoder function automatically generates a value with the help of the presented function.

external function f\_crypt\_password(in octetstring P,in octetstring req\_auth,in octetstring salt,in boolean decrypt,in charstring secret) return octetstring;

This function calculates the MD5 hash in Attribute User-Password for outgoing and incoming messages according to RFC 2865, Section 5.2.

external function f\_crypt\_s\_key(in octetstring pl\_s\_key,in octetstring pl\_req\_auth,in charstring secret,in boolean decrypt) return octetstring;

This function calculates the MD5 hash in cdma2000 subattributum s\_key for outgoing and incoming messages.

external function f\_crypt\_tunnel\_password(in octetstring P,in octetstring req\_auth,in octetstring salt,in boolean decrypt,in charstring secret) return octetstring;

This function calculates the MD5 hash in Attribute Tunnel-Password for outgoing and incoming messages according to RFC 2868, Section 3.5.

EAP functions:

The following functions are available in PPP\_CNL113599 Protocol Module to encode and decode EAP Packets which can be encapsulated in EAP-Message Attribute (Type: 79)

Functions for encode and decode concatenated EAP PDUs:

external function f\_enc\_PDU\_EAP\_list(in PDU\_EAP\_list pdu) return octetstring;

external function f\_dec\_PDU\_EAP\_list(in octetstring stream) return PDU\_EAP\_list;

Functions for encode and decode eap\_sim\_attrib\_list and eap\_aka\_attrib\_list, these kinds of lists can be encrypted in *AT\_ENCR\_DATA*:

external function f\_enc\_eap\_sim\_attrib\_list(in eap\_sim\_attrib\_list pdu) return octetstring

external function f\_dec\_eap\_sim\_attrib\_list(in octetstring stream) return eap\_sim\_attrib\_list

external function f\_enc\_eap\_aka\_attrib\_list(in eap\_aka\_attrib\_list pdu) return octetstring

external function f\_dec\_eap\_aka\_attrib\_list(in octetstring stream) return eap\_aka\_attrib\_list

When using EAP messages the value of the *AT\_MAC* attribute can be set to zero. In this case it will be calculated automatically. It is possible to put a nonzero value directly into this field. In this case it will be sent without any changes to the remote host. The PPP protocol module supports MAC calculation:

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

Function for initialize port descriptor variable The initialization is very important, the use of a descriptor variable without initialization can occur errors!!!:

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

Using the following function authentication and encryption key generation is automatic:

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

Since the *K*i key in EAP SIM and *K* key in EAP AKA also contributes to *Kencr* it has to be set from the test suite prior to sending or receiving EAP-SIM messages containing *AT\_ENCR\_DATA* using the following function:

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

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

Before sending *AT\_AUTN* attribute it is necessary to set *SQN* (6 octet long) and *AMF* (2 octet long) values and before sending *AT\_AUTS* attribute it is necessary to set *SQN\_MS* value. The following external functions are available to set these attributes:

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

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

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

Authentication and encryption key generating functions:

SIM key calculation:

Function for calculating *n\*SRES*:

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

Function for calculating *n\*Kc* from *Ki* and *AT\_RAND*:

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

Function for calculating *Kaut* and *Kenc* from the concatenated input value (*Identity*|*n\*Kc*|*NONCE\_MT*|*Version List*|*Selected Version*):

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

AKA key calculation:

The following function calculates the *K\_aut*, *K\_enc*, *AK* values and returns with *XDOUT*. *K\_aut* is used when calculating *MAC* values, and *K\_encr* 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.

external 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

Functions for encrypting and decrypting SIM and AKA *AT\_ENCR\_DATA*:

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

external 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;

external 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;

Vendor specific attribute handling functions:

Encode/decode salted attribute:

external function f\_salt\_value(inout vendor\_specific\_value pdu, in octetstring req\_auth, in charstring secret, in boolean decrypt) return boolean;

Convert the value of the attribute into text if possible:

external function f\_convert\_string\_to\_text(inout vendor\_specific\_value pdu) return boolean;

# Upgrading templates used by the RADIUS test port

The RPMG type structure differs from the one that is used in the RADIUS message test port [5]. This causes backward incompatibilities in the TTCN-3 type definition module. Therefore, functions and templates developed for RADIUS message test port [5] need to be updated according to the changes of the type definition so that they can be used with RPMG.

In case new fields were added into existing record or set types, the new templates should contain these fields set to omit.

In case a type has changed completely the whole template or part of template must be changed.

If a function is accessing a field that has changed that function needs to be updated as well.

Here you can find a list of major changes:

1. The module name containing RADIUS type definitions has been changed, thus you should replace RADIUSmsg\_Types by RADIUS\_Types in import lines of modules using RADIUS type definitions.
2. The name of top level PDU changed from RADIUS\_message to PDU\_RADIUS.
3. The enumerated type that contains message types is renamed from message\_type to Code, and the names of enumerated items are capitalized.
4. The enumerated type that contains Attribute types is renamed from attrib\_type\_enum to Attrib, and the names of enumerated items are changed according to the naming convention of the protocol module generator (see clause 3.3).
5. The name of the third field of an Attribute record depends on the name of the Attribute instead of its type. For example base\_user\_name is used instead of string\_val.
6. The union type RADIUS\_attrib was modified. Its new name is GenericAttrib. A GenericAttrib type was introduced for error-handling purposes. Its entire branch contains the correctly decoded Attributes, excepting the last one. If something goes wrong during decoding, the attrib\_UNKNOWN branch is used, which is of type record.
7. The field names of the union type GenericAttrib are denominated according to the naming convention (see clause 3.3).
8. All type names of Attributes are changed according to the naming convention (see clause 3.3).
9. Name of enumeration types and values within Attributes are changed according to 3.3.2.

# Error handling

During the encoding and the decoding of a RADIUS message the following error scenarios can be identified:

* If a RADIUS message arrives with a code not known by the RADIUS protocol module, an error message is generated, where the unknown code value appears at <value>: Warning: While RAW-decoding type '@RADIUS\_Types.PDU\_RADIUS': Invalid enum value <value> for '@RADIUS\_Types.Code'
* If the Code field of the RADIUS message to be sent is invalid, an error message is generated, where the code value appears at <value>: Warning : Will not send out Radius message with code <value> in Server/Client mode!
* If a RADIUS message arrives with a wrong authentication value, an error message is generated, where the code value appears at <value>: Warning: Unexpected authentication value received in message with code <value>
* In case of an unknown Attribute code, the Attribute is decoded into a special attrib\_UNKNOWN field. The type of this field is Attrib\_ UNKNOWN. That is a record, which contains three fields (attrib\_type, attrib\_length, attrib\_value) like a normal Attribute with octetstring-type attrib\_value.

# Examples

The “demo” directory of the deliverable contains examples (RADIUS\_Test\_orig.ttcn, RADIUS\_Test\_new.ttcn, RadiusAndEAP\_Demo.ttcn) and reusable modules (RADIUS\_Mapping\_orig.ttcn, RADIUS\_Mapping\_new.ttcn, RADIUSandEAP\_Mapping.ttcn) for RPMG.

## Mapping module

The RADIUS\_Mapping.ttcn (RADIUS\_Mapping\_orig.ttcn, RADIUS\_Mapping\_new.ttcn, RADIUSandEAP\_Mapping.ttcn) module provides the connection between the RADIUS protocol module and the UDP (CNL 113 346) test port. It maintains UDP connections and encodes/decodes RADIUS messages.

The mapping component supports client and server mode operations and sends notifications about the state of the underlying UDP connections to the mapping users.

The module automatically encrypts the User-Password attribute’s String with MD5. It means that the base\_user\_password field of attrib\_Base\_User\_Password attribute in TTCN-3 source code has to be in OCTETSTRING format without any MD5 encryption. Note that the char2oct() function in TTCN-3 converts the password given in ASCII to OCTETSTRING.

The mapping module calculates the Request Authenticator field and checks the Response Authenticator field.

Note that if the Request Authenticator field is zero then the mapping module will calculate automatically the MD5 encrypted checksum. It is possible to put a nonzero value directly into this field. In this case the test port will send this value without any changes to the remote host.

In case of incoming response messages the mapping module checks the Response Authenticator field. If any error is detected then the warning message ‘Unexpected authentication value received in message with code <value>’ appears, where the code value appears at <value>.

#### Configuration

The following module parameter is used also in client and server mode:

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type** | **Description** |
| tsp\_SharedSecret | charstring | Secret shared between the client and the RADIUS server. It should be given in clear text. It is used to calculate and check the authenticator field |

The following module parameters are used when PPP EAP is imported:

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type** | **Description** |
| tsp\_SIM\_Ki | OCT16 | The AKA SQN key will be set automatically with tsp\_SIM\_Ki if it is not set with f\_set\_Ki function. |
| tsp\_AKA\_K | OCT16 | The AKA SQN key will be set automatically with tsp\_AKA\_K if it is not set with f\_set\_K function. |
| tsp\_AKA\_SQN | OCT6 | The AKA SQN key will be set automatically with tsp\_AKA\_SQN if it is not set with f\_set\_SQN function. |
| tsp\_AKA\_SQN\_MS | OCT6 | The AKA SQN key will be set automatically with tsp\_AKA\_SQN\_MS if it is not set with f\_set\_SQN\_MS function. |
| tsp\_AKA\_AMF | OCT2 | The AKA SQN key will be set automatically with tsp\_AKA\_AMF if it is not set with f\_set\_AMF function. |
| tsp\_skip\_auth\_encr | boolean | If this parameter is set to “true”, the authentication and encryption functionality of EAP-SIM and EAP-AKA is turned off. |
| tsp\_global\_keying | boolean | If this parameter is set to “true”, then the test port uses global keying with EAP-SIM and EAP-AKA. |
| tsp\_debugging | boolean | This parameter can turn on and off some verbose logging. |

### Client mode

#### Overview



Figure 2 Client mode mapping

In client mode the RADIUS\_Mapping\_CT initiates connection to the destination host using the UDP\_PCO port. Several users may connect to the mapping component (Figure 2). The users can send PDU\_RADIUS messages to the mapping component, which will be encoded and will be sent through the UDP\_PCO port. The mapping component keeps track of the user\_password attribute and authenticator field of each RADIUS message. The corresponding answers are routed back to the originating user.

The above-described functionality is implemented in separate functions of the RADIUS\_Mapping.ttcn module:

1. f\_handle\_outgoing\_client\_message()
2. f\_handle\_incoming\_client\_message()

### Server mode



Figure 3 Server mode mapping

In server mode the RADIUS\_Mapping\_CT starts listening on a configured port using the UDP\_PCO port and waits for incoming connections. Only one user component shall connect to the mapping component (Figure 3). The user component can send/receive RADIUS messages (ASP\_UDP\_RADIUS).

The above-described functionality is implemented in separate functions of the RADIUS\_Mapping.ttcn module:

1. f\_handle\_outgoing\_server\_message()
2. f\_handle\_incoming\_server\_message()

#### Configuration

The UDP test port must be configured in the following way:

* localPort contains the port number of the server
* Other UDP test port parameters should not be used.

Note that, in case of UDP the listening interface must be given using the localPort UDP test port parameter.

### Attributes of the RADIUSmsg\_PT port

The users can connect to the mapping component via a RADIUSmsg\_PT (RADIUS\_PCO) port. This port conveys the following messages and Attributes:

* **PDU\_RADIUS**: This type contains the RADIUS message representation in TTCN-3
* **ASP\_UDP\_RADIUS**: This type has three fields:
  + *data*: Its type is PDU\_RADIUS and contains a RADIUS PDU
  + *addressf*: Its type is AddressType. This field contains the remote host IP address.
  + *portf*: Its type is PortType. This field contains the remote host port number.

## Demo module

### Test cases

In the demo directory there are test modules (Radius\_Test\_orig.ttcn for original style structured code, Radius\_Test\_new.ttcn for new style structured code and RadiusAndEAP\_Demo.ttcn). These contain example testcases with their used templates, to show how the templates based on RADIUS type definitions look like, and how to start and use the mapping module. The following testcases demonstrate client and server mode operation:

* tc\_RADIUS\_client\_sendAccessRequest()

send Access Request packet

* tc\_RADIUS\_client\_sendAccessRequest\_receiveAccessAccept()

send Access Request packet and receive Access Accept packet

* tc\_RADIUS\_client\_sendAccessAccept()

try to send Access Accept packet

* tc\_RADIUS\_client\_sendAccountingRequest\_receiveAccountingResponse()

send Accounting Request packet and receive Accounting Response packet

* tc\_RADIUS\_server\_receiveAccessRequest\_sendAccessAccept()

receive Access Request packet and send Access Accept packet

* tc\_RADIUS\_server\_receiveAccessRequest\_sendAccessChallenge()

receive Access Request packet and send Access Challenge packet

* tc\_RADIUS\_server\_receiveAccountingRequest\_sendAccountingResponse()  
    
  receive Accounting Request packet and send Accounting Response packet
* tc\_RADIUS\_server\_receiveAccessRequest\_sendAccessAccept\_with\_correct\_auth()

receive Access Request packet and send Access Accept packet with correct authentication

* tc\_RADIUS\_server\_receiveAccessRequest\_sendAccessAccept\_with\_incorrect\_auth()

receive Access Request packet and send Access Accept packet with incorrect authentication

The testcases in RadiusAndEAP\_Demo.ttcn demonstrate EAP SIM and AKA message handling.

* tc\_RADIUS\_EAP\_AKA\_full\_authentication
* tc\_RADIUS\_EAP\_SIM\_full\_authentication
* tc\_RADIUS\_EAP\_SIM\_reauthentication

### Configuration files

There is example configuration file in the demo directory as well, that can be used when executing the example test cases:

* radius.cfg
* RADIUSandEAP.cfg

### Examples for building the project

For GUI users there are files as examples (RADIUS\_Demo\_orig.prj for original style structured code, RADIUS\_Demo\_new.prj for new style structured code and RADIUSandEAP\_Demo.prj). Do not forget to generate softlinks for the files under the ‘Misc Files’ section (see clause 5.3 for more details).

### Script to modify Makefile

In the demo directory there are example shell scripts (gui\_make\_RPGM\_orig.sh for original style structured code, gui\_make\_RPGM\_new.sh for new style structured code and gui\_make\_RPMGandEAP.sh) to modify the generated makefile. These scripts can be used by the GUI.

Example (gui\_make\_RPGM\_new.sh):

#!/bin/sh

sed -e '

s/OPENSSL\_DIR = $(TTCN3\_DIR)/\OPENSSL\_DIR = \/mnt\/TTCN\/Tools\/openssl-0.9.8e/g

s/CPPFLAGS = -D$(PLATFORM) -I$(TTCN3\_DIR)\/include/CPPFLAGS = -D$(PLATFORM) -I$(OPENSSL\_DIR)\/include -I$(TTCN3\_DIR)\/include/g

' \

-e 's/^TTCN3\_MODULES =/TTCN3\_MODULES = RADIUS\_Types.ttcn/g

s/^GENERATED\_SOURCES =/GENERATED\_SOURCES = RADIUS\_Types.cc/g

s/^GENERATED\_HEADERS =/GENERATED\_HEADERS = RADIUS\_Types.hh/g

s/^OBJECTS =/OBJECTS = RADIUS\_Types.o/g

/# Add your rules here if necessary./ {

a\

#

a\

a\

AWK=/usr/local/bin/gawk

a\

a\

RADIUS\_Types.ttcn: RadiusBaseTypes.rdf RadiusBaseAttributes.rdf RadiusAccountingAttributes.rdf RadiusAndIPv6.rdf RadiusExtensions.rdf TunnelAuthenticationAttributes.rdf ATTR.awk

a\

$(AWK) -f ATTR.awk RadiusBaseTypes.rdf RadiusBaseAttributes.rdf RadiusAccountingAttributes.rdf RadiusAndIPv6.rdf RadiusExtensions.rdf TunnelAuthenticationAttributes.rdf > $@

a\

a\

#

a\

# End of additional rules for RPMG

}

' \

<$1 >$2

# Terminology

RDF RADIUS Definitions File: TTCN-3 type definitions describing RADIUS Attributes outside module

RPMG RADIUS Protocol Module Generator

## Abbreviations

TCC Test Competence Center

TTCN-3 Testing and Test Control Notation version 3

RADIUS Remote Authentication Dial In User Service

MD5 Message-Digest Algorithm 5

ASP Abstract Service Primitive

RPMG RADIUS Protocol Module Generator

GUI Graphical User Interface

PDU Protocol Data Unit

SUT System Under Test

UDP User Datagram Protocol

# References

1. RFC 2865  
   Remote Authentication Dial In User Service (RADIUS)
2. ETSI ES 201 873-1 v3.2.1 (2007-02)  
   The Testing and Test Control Notation version 3. Part 1: Core Language
3. 109 21-CNL 113 600-11  
   RADIUS Protocol Module Generator for TTCN-3 Toolset with TITAN, Product Revision Information
4. 1/198 17-CRL 113 200/4 Uen  
   Programmer’s Technical Reference for the TITAN TTCN-3 Test Executor
5. 109 21-CNL 113 311-15 Uen   
   RADIUS Test Port for TTCN-3 Toolset with TITAN, Product Revision Information
6. The GNU Awk User's Guide, <http://www.gnu.org/software/gawk/manual/gawk.html>
7. RFC 2866  
   RADIUS Accounting
8. RFC 2868  
   RADIUS Attributes for Tunnel Protocol Support
9. RFC 2869  
   RADIUS Extensions
10. RFC 3162  
    RADIUS and IPv6
11. RFC 5176  
    RADIUS Authorization Extensions
12. RFC 4372  
    Chargeable User Identity
13. 46/155 17-CSA 113 35/4 Uen Rev B  
    GGSN Functional Specification: RADIUS
14. RFC 2867  
    RADIUS Accounting Modifications for Tunnel Protocol Support
15. EEM/T-12:0148 Rev PA5

RADIUS Interface Description for Ericsson

Network Integrated WiFi (ENIW) solution

SAPC - WiFi Gateway

1. 1/1531-CRL 113 200/4 Uen  
   Installation Guide for the TITAN TTCN-3 Test Executor
2. 198 17-CNL 113 346 Uen  
   UDP Socket Test Port for TTCN-3 Toolset with TITAN, UG
3. OpenSSL toolkit  
   <http://www.openssl.org>

# Change information

## R6A

CR\_TR00019869 has been implemented.

## R9A

Artf372817 has been implemented.

## R10A

The AVPs from VCPE.rdf was added to Smartedge.rdf

## R11A

[Artf388130](https://eforge.ericsson.se/sf/go/artf388130) has been implemented.