Radiator® SIM Module

Installation and reference manual for Radiator® SIM Module 2.7. Last revised on November 18, 2021

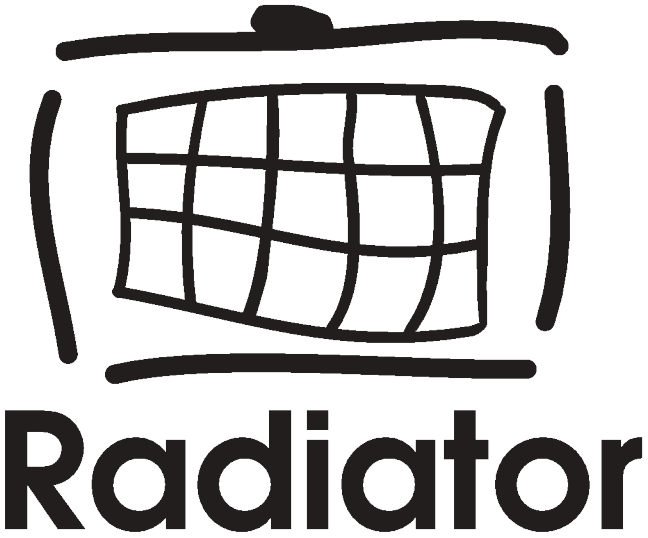

Table of Contents

1. Introduction to Radiator SIM Module

This document describes how to install and configure the Radiator SIM Module.

Radiator SIM Module provides EAP-SIM (Extensible Authentication Protocol - Subscriber Identity Module), EAP-AKA (Extensible Authentication Protocol - Authentication and Key Agreement), and EAP-AKA' (Extensible Authentication Protocol - Authentication and Key Agreement Prime) support for Radiator RADIUS server. RADIUS is the de facto standard protocol for authenticating users and recording accounting information for wired and wireless LAN (Local Area Network)s. It is supported and used by most vendors, such as Cisco, Ericsson, Huawei, Juniper, Ruckus, Aruba, and Alcatel-Lucent.

Radiator SIM Module includes the following features:

- 3GPP AAA (Authentication, Authorisation, Accounting) Server SWm, S6b, SWa, and STa Diameter interfaces

  For more information about Radiator 3GPP AAA Server, see Radiator 3GPP AAA reference manual [https://radiatorsoftware.com/products/radiator-sim-pack/].

- IMSI privacy as defined by 3GPP document S3-170116 and Wireless Broadband Alliance technical specification 'IMSI Privacy Protection for Wi-Fi'.
- 3GPP Diameter Wx interface for direct HSS (Home Subscriber Server) connections
- 3GPP Diameter SWx interface for direct HSS connections
- M3UA (MTP Level 3 User Adaptation Layer)/SIGTRAN interface for MAP (Mobile Application Part)-based authentication
- Support for converting AKA (Authentication and Key Agreement) vectors to SIM (Subscriber Identity Module) triplets
- Support for MAP/SS7 (Signalling System No. 7) gateways for connecting to legacy AuC (Authentication Centre)/HLR (Home Location Register) servers over MAP, SIGTRAN, and other protocols
- Support for full RADIUS accounting
- Support for both IPv4 and IPv6
- TCP (Transmission Control Protocol) and SCTP (Stream Control Transmission Protocol) for both IPv4 and IPv6

EAP-SIM is an EAP (Extensible Authentication Protocol) authentication protocol, designed to be used with existing GSM mobile phone authentication systems and SIM cards for mobile phones. The EAP-SIM standard allows Wireless LAN users to authenticate access to a Wireless LAN network using a mobile phone SIM card.

EAP-AKA is an EAP authentication protocol for authentication and session key distribution. It is based to the AKA mechanism: challenge-response and symmetric cryptography. EAP-AKA' is a variant of EAP-AKA, it is used for accessing 3GPP services via non-3GPP access.

2. Installing Radiator SIM Module

This section the describes how to install Radiator SIM Module.

If TMSI (Temporary Mobile Subscriber Identity) or Fast Re-Authentication support is required, you need an SQL database with sample tables created from `goodies/eap-sim-mysql.sql`. For MySQL or MariaDB, `DBD::mysql` is required. Other types of databases and alternative database schemas are also supported. For more information, contact Radiator Software.

2.1. Prerequisites

The prerequisites for Radiator 3GPP AAA Server are described in its reference manual. To be able to install Radiator SIM module, your system must meet these prerequisites

- Radiator 4.26 or later is recommended. Absolute minimum is Radiator 4.14.

  If you need *`LocalAddress`* or *`LocalPort`* parameters, you must have Radiator 4.17 or later.

- Radiator Carrier module 1.7 or later is recommended.
- `Radius::UtilXS` 2.2 or later is recommended. Otherwise additional modules from CPAN, as shown below, are needed.
- SCTP multihoming requires Radiator `Radius::UtilXS` module.
- IMSI encryption requires Radiator `Radius::UtilXS` module 2.0 or later.
- The following Perl modules:
  - `Radius::UtilXS` 2.2 or later - From Radiator downloads, or `Crypt::Rijndael`
  - `DBI`
  - `Radius::UtilXS` 2.0 or later - From Radiator downloads, or `Digest::SHA1`
  - `Digest::SHA` - Typically already installed with system Perl
  - `Convert::ASN1` version 0.26 or later, if you need SIGTRAN *<PeerSP>* support

2.2. Installation

Procedure

The recommended method is to install Radiator, Radiator Carrier and Radiator SIM Module from operating system specific packages. If required, source code installation is also possible. Operating system specific packages are available as stand-alone downloads and as repositories that integrate with operating system package management tools, such as **`yum`** and **`apt`**. See Radiator download site for detailed repository instructions.

To install Radiator SIM Module using stand-alone package:

1. Download the Radiator SIM Module distribution.
2. Unpack the file into a separate working directory.
3. Move to the distribution directory.
4. Prepare the distribution for installation.

   ```
   perl Makefile.PL
   ```

5. Run the installation. You may need the root access rights for running this command.

   ```
   make install
   ```

6. Build a Radiator configuration file based on one of the following configuration files:
   - `goodies/eap_sim_wx.cfg`
   - `goodies/eap_aka_wx.cfg`
   - `goodies/eap_sim_map.cfg`
   - `goodies/eap_aka_map.cfg`

7. Configure Radiator HSS emulator file (`goodies/wxmap.cfg`), or connect to the HSS over Wx or SWx, or connect to the HLR over SIGTRAN.
8. Run Radiator with the configuration file developed in the step 6.
9. Set Radiator to start automatically when booting. For more information, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf].

3. Testing Radiator SIM Module

This section describes the test scenarios for Radiator SIM Module and how to compile `eapol_test` for testing different EAP protocols without the actual infrastructure.

3.1. Compiling `eapol_test`

`eapol_test` is a part of wpa_supplicant suite [http://w1.fi/wpa_supplicant/]. It is a tool for testing Radiator EAP-SIM, EAP-AKA, and EAP-AKA' protocols. You can configure it to act as a supplicant to generate RADIUS requests which are sent directly to the RADIUS server. With `eapol_test`, you can test the system without the client, supplicant, and wireless access point.

Note

The `eapol_test` configuration `.config` file located in `wpa_supplicant2.4/wpa_supplicant/`. After configuring it, always rerun `make eapol_test` because the `eapol_test` target is not a part of the default `make` target.

Enabling SIM method

EAP-SIM is not enabled by default. If you try to test it when it is disabled, the following error occurs:

```
Line 19: unknown EAP method 'SIM'
```

```
You may need to add support for this EAP method during wpa_supplicant build time configuration.
```

```
See README for more information.
```

To enable EAP-SIM, add *`CONFIG_EAP_SIM`* to `.config` file:

```
echo CONFIG_EAP_SIM=y >> .config make eapol_test
```

Enabling Milenage SIM emulator for EAP-SIM

To be able to use format *`password="Ki:OPc"`* in `eapol_test .config` file, `eapol_test` must be compiled with the internal GSM-Milenage implementation. If it is not compiled, the authentication process gives the following error message:

```
EAP-SIM: 3 challenges
```

```
EAP-SIM: GSM authentication algorithm
```

```
EAP-SIM: No GSM authentication algorithm enabled
```

```
EAP-SIM: GSM authentication failed
```

```
EAP-SIM: CONTINUE -> FAILURE
```

To compile the `eapol_test` with the internal GSM-Milenage implementation:

```
echo CONFIG_SIM_SIMULATOR=y >> .config make eapol_test
```

Enabling AKA methods and USIM simulator

For EAP-AKA and EAP-AKA' the Milenage parameters are defined in format **`password="Ki:OPc:SQN"`** in `eapol_test .config` file.

To enable the AKA methods and USIM (Universal Subscriber Identity Module) simulator:

```
echo CONFIG_EAP_AKA=y >> .config
```

```
echo CONFIG_EAP_AKA_PRIME=y >> .config
```

```
echo CONFIG_USIM_SIMULATOR=y >> .config
```

```
make eapol_test
```

3.2. Testing EAP-SIM with unknown SIM card and canned triplets

About this task

This test scenario tests EAP-SIM functionality when the SIM card's secret keys are not known. The SIM triplets are generated by the SIM card and added to a triplet file. Radiator SWx/Wx Diameter server is configured to use the triplet file for authentication.

Before you begin

You need the following accessories and softwares for this test scenario:

- PCSC (Personal Computer Smart Card) reader
- pcsc-lite, PCSC smart card software
- pcsc-perl, Perl wrapper for PCSC
- GSM SIM cards for testing

Procedure

To execute the test:

1. If you use TMSI or Fast Re-Authentication, create a sample SQL tables into the SQL database.

   ```
   mysql -usqluser -psqlpassword databasename <goodies/eap-sim-mysql.sql
   ```

   - *`sqluser`* is the MySQL user name.
   - *`sqlpassword`* is the MySQL password.
   - *`databasename`* is the name of the test MySQL database.
2. Generate triplets with the test GSM SIM card and add them to the triplet file. Use the correct PIN.

   ```
   goodies/gettriplets -pin 1234 >>goodies/triplets.dat
   ```
3. Specify the triplets file path in `goodies/wxmap.cfg`.

   ```
   TripletsFile path/to/your/goodies/triplets.dat
   ```
4. Run the Radiator SWx/Wx server.

   ```
   radiusd -config ../Radius-EAP-SIM/goodies/wxmap.cfg
   ```
5. Run the Radiator EAP server in another window.

   ```
   radiusd -config goodies/eap_sim_wx.cfg
   ```
6. Insert the test SIM card to the EAP-SIM supplicant and create a test connection.

7. For TMSI or Fast Re-Authentication support, enable *UseTMSI* on page 8 and *UseReauthentication* on page 8 flags in the Radiator EAP server configuration file.

3.3. Testing EAP-SIM with known Milenage SIM card

About this task

This test scenario tests EAP-SIM functionality when the SIM card has the Milenage key algorithm and also the secret keys are known.

Before you begin

You need a SIM card for which you know the embedded Milenage keys. Alternatively you can use *wpa_supplicant* or *eapol_test* with a card simulator to specify the Milenage secret keys.

Procedure

To execute the test:

1. If you use TMSI or Fast Re-Authentication, create a sample SQL tables into the SQL database.

   ```
   mysql -usqluser -psqlpassword databasename <goodies/eap-sim-mysql.sql
   ```

   - *sqluser* is the MySQL user name.
   - *sqlpassword* is the MySQL password.
   - *databasename* is the name of the test MySQL database.

2. Add the Milenage key data to the card data file `goodies/simcards.dat` by adding a new line to the file and specify the card keys in the following format:

   IMSI (International mobile subscriber identity) Ki (Authentication key) OPc (Operator Code) AMF (Authentication Management Field) SQN (Sequence Number)

3. Specify the card credentials file path in `goodies/wxmap.cfg`.

   ```
   3GPPCardDatabaseFilename path/to/your/goodies/simcards.dat
   ```

4. Run the Radiator SWx/Wx server.

   ```
   radiusd -config ../Radius-EAP-SIM/goodies/wxmap.cfg
   ```

5. Run the Radiator EAP server in another window.

   ```
   radiusd -config goodies/eap_sim_wx.cfg
   ```

6. Insert the test SIM card to the EAP-SIM supplicant and create a test connection.

   Alternatively you can run *wpa_supplicant* or *eapol_test* with a configuration that simulates the card.

   ```
   ./eapol_test -p 1645 -s mysecret -c path/to/your/goodies/sim-simulator.conf
   ```

7. For TMSI or Fast Re-Authentication support, enable *UseTMSI* on page 8 and *UseReauthentication* on page 8 flags in the Radiator EAP server configuration file.

3.4. Testing EAP-AKA with known Milenage 3G/LTE SIM card

About this task

This test scenario tests EAP-AKA functionality when the 3G/LTE SIM card has the Milenage key algorithm and also the secret keys are known.

Before you begin

You need a 3G/LTE SIM card for which you know the embedded Milenage keys. Alternatively you can use *wpa_supplicant* or *eapol_test* with a card simulator to specify the Milenage secret keys.

Procedure

To execute the test:

1. If you use TMSI or Fast Re-Authentication, create a sample SQL tables into the SQL database.

   ```
   mysql -usqluser -psqlpassword databasename <goodies/eap-sim-mysql.sql
   ```

   - *sqluser* is the MySQL user name.
   - *sqlpassword* is the MySQL password.
   - *databasename* is the name of the test MySQL database.

2. Add the Milenage key data to the card data file `goodies/simcards.dat` by adding a new line to the file and specify the card keys in the following format:

   IMSI Ki OPc AMF SQN

3. Specify the card credentials file path in `goodies/wxmap.cfg`.

   ```
   3GPPCardDatabaseFilename path/to/your/goodies/simcards.dat
   ```

4. Run the Radiator SWx/Wx server.

   ```
   radiusd -config ../Radius-EAP-SIM/goodies/wxmap.cfg
   ```

5. Run the Radiator EAP-SIM server in another window.

   ```
   radiusd -config goodies/eap_aka_wx.cfg
   ```

6. Insert the test SIM card to the EAP-AKA supplicant and create a test connection.

   Alternatively you can run *wpa_supplicant* or *eapol_test* with a configuration that simulates the card.

   ```
   ./eapol_test -p 1645 -s mysecret -c path/to/your/goodies/aka-simulator.conf
   ```

7. For TMSI or Fast Re-Authentication support, enable *UseTMSI* on page 8 and *UseReauthentication* on page 8 flags in the Radiator EAP server configuration file.

3.5. Testing EAP-AKA' with known Milenage 3G/LTE SIM card

About this task

This test scenario tests EAP-AKA' functionality when the 3G/LTE SIM card has the Milenage key algorithm and also the secret keys are known.

Before you begin

You need a 3G/LTE SIM card for which you know the embedded Milenage keys. Alternatively you can use *wpa_supplicant* or *eapol_test* with a card simulator to specify the Milenage secret keys.

Procedure

To execute the test:

1. If you use TMSI or Fast Re-Authentication, create a sample SQL tables into the SQL database.

   ```
   mysql -usqluser -psqlpassword databasename <goodies/eap-sim-mysql.sql
   ```

- *sqluser* is the MySQL user name.
- *sqlpassword* is the MySQL password.
- *databasename* is the name of the test MySQL database.

2. Add the Milenage key data to the card data file `goodies/simcards.dat` by adding a new line to the file and specify the card keys in the following format:

   IMSI Ki OPc AMF SQN

3. Specify the card credit file path in `goodies/wxmap.cfg`.

```
3GPPCardDatabaseFilename path/to/your/goodies/simcards.dat
```

4. Run the Radiator SWx/Wx server.

   `radiusd -config ../Radius-EAP-SIM/goodies/wxmap.cfg`

5. Run the Radiator EAP server in another window.

   `radiusd -config goodies/eap_aka_prime_wx.cfg`

6. Insert the test SIM card to the EAP-AKA' supplicant and create a test connection.

   Alternatively you can run *wpa_supplicant* or *eapol_test* with a configuration that simulates the card:

   `./eapol_test -p 1645 -s mysecret -c path/to/your/goodies/aka-prime-simulator.conf`

7. For TMSI or Fast Re-Authentication support, enable *UseTMSI* on page 8 and *UseReauthentication* on page 8 flags in the Radiator EAP server configuration file.

4. Configuring Radiator SIM Module

This section describes the configurable parameters of Radiator SIM Module.

4.1. <AuthBy AKA>

This section describes the configuring parameters of *<AuthBy AKA>*. Apart from the parameters listed here, *<AuthBy AKA>* inherits other parameters from *AuthGeneric*. These parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section <AuthBy xxxxxx>.

4.1.1. AKAPrimeKDF

This string defines the KDF (Key Derivation Function) algorithm used for AKA' (Authentication and Key Agreement Prime). The default value is **`DEFAULT`**, which means the default AKA' KDF.

4.1.2. AKAPrimeNetworkName

This string defines the network name used for AKA' authentication. The default value is **`WLAN`**.

4.1.3. AuthorisedHook

This is a Perl hook, which is called when the client is authenticated and authorised, before the **`EAP Success`** message is sent. *AuthorisedHook* is called with 5 arguments:

- **`$_[0]`**

  This is a pointer to this *AuthBy*.

- **`$_[1]`**

  This is a pointer to the current EAP context structure of the current user.

- **`$_[2]`**

  This is a pointer to the last request from the client.

- **`$_[3]`**

  This is a pointer to the authentication result. It is preset to **`main::ACCEPT`** but it can be changed if needed.

- **`$_[4]`**

  This is a pointer to the authentication reason. It is preset to empty string but it can be changed if needed.

4.1.4. MaxReauthentications

This integer defines the maximum number of permitted consecutive reauthentications. The default value is **`100`**.

4.1.5. ReauthenticationRealm

This string defines if *`ReauthenticationRealm`* is appended to the reauthentication ID. It ensures that RADIUS routing gets the reauthentication request back to the server, which handles the request. This has no default value.

The following formatter is available in *`ReauthenticationRealm`*:

- *`%0`*

  This is the realm part of identity.

For more information about formatters, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section Special characters.

4.1.6. ResLengthInBytes

This flag allows interoperation with the old AKA clients, such as *`wpa_supplicant`* up to at least version 0.6.5, but not with version 0.6.9 or later, which use an obsolete format for *`AT_RES`* encoding. This is now obsolete and should not be used. This is not set by default.

4.1.7. UseReauthentication

This flag defines if the server offers and accepts the reauthentication IDs. This is not set by default.

4.1.8. UseResultInd

This flag enables the *`AT_RESULT_IND`* usage. *`AT_RESULT_IND`* is a flag which, when set, requires result indications if they are also enabled in the peer. This is not set by default.

4.1.9. UseTMSI

This flag defines whether the server offers and accepts pseudonyms. This is not set by default.

4.1.10. IMSICrypt

This string defines the identifier of *`IMSICrypt`* clause to use for IMSI decryption. For more about IMSI encryption, see Section 4.17. <IMSICrypt> on page 34.

4.2. <AuthBy AKAMAP>

This section describes the configuring parameters of *<AuthBy AKAMAP>*. Apart from the parameters listed here, *<AuthBy AKAMAP>* inherits other parameters from *<AuthBy AKASQL>* on page 9.

4.2.1. MAP

This object list lists the *<MAP>* on page 26 clauses used by this *<AuthBy SIMMAP>*.

4.3. <AuthBy AKAREST>

This section describes the configuration parameters of *<AuthBy AKAREST>*. Apart from the parameters listed here, *<AuthBy AKAMAP>* inherits other parameters from *<AuthBy AKASQL>* on page 9.

REST interface is provided by Radiator's HTTP client module. HTTP client parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section *HTTP client configuration*.

<AuthBy AKAREST> fetches authentication vectors over a HTTP or HTTPS REST interface. See `goodies/eap_aka_rest.cfg` and `goodies/eap_aka_prime_rest.cfg` for a sample configurations. The REST API is documented in `goodies/rest.txt`.

4.3.1. NoReplyReject

When set, this optional flag parameter causes Radiator to REJECT the request instead of returning IGNORE if badly formatted REST reply or no RESTreply at all is received. This parameter is not set by default.

4.4. <AuthBy AKASQL>

This section describes the configuring parameters of *<AuthBy AKASQL>*. Apart from the parameters listed here, *<AuthBy AKASQL>* inherits other parameters from *<AuthBy AKA>* on page 7 and Radiator SQL database module. The inherited SQL database parameters are:

- *ConnectionAttemptFailedHook*
- *ConnectionHook*
- *DateFormat*
- *DBAuth*
- *DBSource*
- *DBUsername*
- *DisconnectAfterQuery*
- *FailureBackoffTime*
- *NoConnectionsHook*
- *RoundRobinOnFailure*
- *SQLRecoveryFile*
- *SQLRetries*
- *Timeout*

These parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section <AuthBy SQL>.

4.4.1. GetTMSIQuery

This string contains the SQL query for mapping the IMSI from a TMSI.

The following bind variable is available in *GetTMSIQuery*:

- *%0*

  This is TMSI.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *GetTMSIQuery*

In the following example query, bind variables marked with question marks are used with *GetTMSIQueryParam* listed below the query.

```
GetTMSIQuery select IMSI from SIMTMSI where TMSI = ?
GetTMSIQueryParam %0
```

Instead of the previous example, you can also use the following query without the *GetTMSIQueryParam*. In this case, SQL quoted value is used to create the SQL query.

```
GetTMSIQuery select IMSI from SIMTMSI where TMSI = %0
```

4.4.2. GetTMSIQueryParam

This string array defines the bind variables to be used with *GetTMSIQuery*. See *GetTMSIQuery* on page 10 for more information about the available bind variables.

4.4.3. SaveTMSIQuery

This string contains the SQL query for saving the IMSI-TMSI mapping.

The following bind variables are available in *SaveTMSIQuery*:

- *%0*

  This is IMSI.

- *%1*

  This is TMSI.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *SaveTMSIQuery*

In the following example query, bind variables marked with question marks are used with *SaveTMSIQueryParam*s listed below the query.

```
SaveTMSIQuery replace SIMTMSI (IMSI, TMSI) values (?, ?)
SaveTMSIQueryParam %0
SaveTMSIQueryParam %1
```

Instead of the previous example, you can also use the following query without the *SaveTMSIQueryParam*. In this case, SQL quoted values are used to create the SQL query.

```
SaveTMSIQuery replace SIMTMSI (IMSI, TMSI) values (%0, %1)
```

4.4.4. SaveTMSIQueryParam

This string array defines the bind variables to be used with *SaveTMSIQuery*. See *SaveTMSIQuery* on page 10 for more information about the available bind variables.

4.4.5. DeleteReauthQuery

This string contains the SQL query for deleting the reauthentication data from the database.

The following bind variable is available in *DeleteReauthQuery*:

- *%0*

  This is the reauthentication ID.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *DeleteReauthQuery* with *<AuthBy AKASQL>*

This example query shows how *DeleteReauthQuery* is used with *<AuthBy AKASQL>*. The bind variables marked with question marks are used with *DeleteReauthQueryParam* listed below the query.

```
DeleteReauthQuery update SIMUSER set  REAUTH_ID=NULL, \
      COUNTER=NULL, NEXT_REAUTH_ID=NULL where REAUTH_ID=?
DeleteReauthQueryParam %0
```

Instead of the previous example, you can also use the following query without the *DeleteReauthQueryParam*. In this case, SQL quoted value is used to create the SQL query.

```
DeleteReauthQuery update SIMUSER set REAUTH_ID=NULL, \
      COUNTER=NULL, NEXT_REAUTH_ID=NULL where REAUTH_ID=%0
```

4.4.6. DeleteReauthQueryParam

This string array defines the bind variables to be used with *DeleteReauthQuery*. See *DeleteReauthQuery* on page 11 for more information about the available bind variables.

4.4.7. GetReauthQuery

This string contains the SQL query for getting the reauthentication data from the database.

The following bind variable is available in *GetReauthQuery*:

- *%0*

  This is TMSI.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *GetReauthQuery* with *<AuthBy AKASQL>*

This example query shows how *GetReauthQuery* is used with *<AuthBy AKASQL>*. In this example, bind variables marked with question marks are used with *GetReauthQueryParam* listed below the query.

```
GetReauthQuery select IMSI, REAUTH_ID, COUNTER, MK, K_AUT, \
```

```
      K_ENCR, K_RE, NEXT_REAUTH_ID, VERSION from SIMUSER \
      where REAUTH_ID = ?
GetReauthQueryParam %0
```

Instead of the previous example, you can also use the following query without the *GetReauthQueryParam*. In this case, SQL quoted value is used to create the SQL query.

```
GetReauthQuery select IMSI, REAUTH_ID, COUNTER, MK, K_AUT, \
      K_ENCR, K_RE, NEXT_REAUTH_ID, VERSION from SIMUSER \
      where REAUTH_ID = %0
```

4.4.8. GetReauthQueryParam

This string array defines the bind variables to be used with *GetReauthQuery*. See *GetReauthQuery* on page 11 for more information about the available bind variables.

4.4.9. SaveReauthQuery

This string contains the SQL query for saving the reauthentication data to the database.

The following bind variables are available in *SaveReauthQuery*:

- *%0*

  This is the reauthentication ID.

- *%1*

  This is the IMSI.

- *%2*

  This is the counter.

- *%3*

  This is the master key.

- *%4*

  This is the authentication key.

- *%5*

  This is the encryption key.

- *%6*

  This is the reauthentication key.

- *%7*

  This is the version of SIM.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *SaveReauthQuery* with *<AuthBy AKASQL>*

This example query shows how *SaveReauthQuery* is used with *<AuthBy AKASQL>*. The bind variables marked with question marks are used with *SaveReauthQueryParam*s listed below the query.

```
SaveReauthQuery replace SIMUSER (IMSI, REAUTH_ID, COUNTER, \
```

```
      MK, K_AUT, K_ENCR, K_RE, VERSION) \
      values (?, ?, ?, ?, ?, ?, ?, ?)
SaveReauthQueryParam %1
SaveReauthQueryParam %0
SaveReauthQueryParam %2
SaveReauthQueryParam %3
SaveReauthQueryParam %4
SaveReauthQueryParam %5
SaveReauthQueryParam %6
SaveReauthQueryParam %7
```

Instead of the previous example, you can also use the following query without the *SaveReauthQueryParam*s. In this case, SQL quoted values are used to create the SQL query.

```
SaveReauthQuery replace SIMUSER (IMSI, REAUTH_ID, COUNTER, \
      MK, K_AUT, K_ENCR, K_RE, VERSION) \
      values (%1, %0, %2, %3, %4, %5, %6, %7)
```

4.4.10. SaveReauthQueryParam

This string array defines the bind variables to be used with *SaveReauthQuery*. See *SaveReauthQuery* on page 19 for more information about the available bind variables.

4.5. <AuthBy AKATEST>

This section describes the configuring parameters of *<AuthBy AKATEST>*. Apart from the parameters listed here, *<AuthBy AKATEST>* inherits other parameters from *<AuthBy AKA>* on page 7.

4.5.1. 3GPPCardDatabaseFilename

This string defines the file path and name where the 2G SIM or 3G USIM card details are stored. Radiator requires read and write access to this file. When defined, this parameter is used to find the Milenage algorithm parameters for SIM and USIM cards. See `goodies/simcards.dat` for a sample file.

The file contains the following information coded to hexadecimal:

- IMSI for the SIM/USIM card
- Ki
- Encrypted OPc
- AMF
- SQN

4.5.2. IndLength

This integer defines the length of the IND part in bits in the AKA authentication vector SQN. The default value is **5**.

4.6. <AuthBy AKAWX>

<AuthBy AKAWX> does not have configurable parameters at the moment. It inherits parameters from *<AuthBy AKASQL>* on page 9 and *<WxClient>* on page 34.

4.7. <AuthBy SIM>

This section describes the configuring parameters of *<AuthBy SIM>*. Apart from the parameters listed here, *<AuthBy SIM>* inherits other parameters from *AuthGeneric*. These parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section <AuthBy xxxxxx>.

4.7.1. AuthorisedHook

This is a Perl hook, which is called when the client is authenticated and authorised, before the **EAP Success** message is sent. *AuthorisedHook* is called with 5 arguments:

- **$_[0]**

  This is a pointer to this *AuthBy*.

- **$_[1]**

  This is a pointer to the current EAP context structure of the current user.

- **$_[2]**

  This is a pointer to the last request from the client.

- **$_[3]**

  This is a pointer to the authentication result. It is preset to **main::ACCEPT** but it can be changed if needed.

- **$_[4]**

  This is a pointer to the authentication reason. It is preset to empty string but it can be changed if needed.

4.7.2. MaxReauthentications

This integer defines the maximum number of permitted consecutive reauthentications. The default value is **100**.

4.7.3. NoSilentDeny

This flag controls the behaviour of EAP-SIM notifications in EAP-SIM version 0. If *NoSilentDeny* is set and the EAP-SIM version number is 0, Radiator sends a notification to the user client if the connection is rejected. If *NoSilentDeny* is not set, no rejection notification is sent. *NoSilentDeny* is not set by default.

4.7.4. NumTriplets

This integer defines the number of triplets requested for SIM authentication. The number must match to what the client is expecting, otherwise the authentication fails. The value is either **2** or **3**, the default value is **3**.

4.7.5. ReauthenticationRealm

This string defines if *ReauthenticationRealm* is appended to the reauthentication ID. It ensures that RADIUS routing gets the reauthentication request back to the server, which handles the request. This has no default value.

The following formatter is available in *ReauthenticationRealm*:

- *%0*

  This is the realm part of identity.

For more information about formatters, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section Special characters.

4.7.6. RequireVersion

This flag defines if the SIM client version is required or not. The recent versions of the EAP-SIM standard require the SIM client to send a protocol version information with each authentication. Some clients (for example, Cisco V5) do not send the protocol version. The default value is **`1`**. Set this parameter to **`0`** only when you must support clients that do not send the version number.

4.7.7. SupportVersions

This string controls which versions of EAP-SIM standard are supported. The parameter is a list of comma-separated version numbers, listed in order of preference, the most preferred first. The default value is **`0,1`** which means that all versions are supported.

4.7.8. UseResultInd

This flag enables the *`AT_RESULT_IND`* usage. *`AT_RESULT_IND`* is a flag which, when set, requires result indications if they are also enabled in the peer. This is not set by default.

4.7.9. UseReauthentication

This flag defines if the server offers and accepts the reauthentication IDs. This is not set by default.

4.7.10. UseTMSI

This flag defines whether the server offers and accepts pseudonyms. This is not set by default.

4.7.11. IMSICrypt

This string defines the identifier of *`IMSICrypt`* clause to use for IMSI decryption. For more about IMSI encryption, see Section 4.17. <IMSICrypt> on page 34.

4.8. <AuthBy SIMMAP>

This section describes the configuring parameters of *`<AuthBy SIMMAP>`*. Apart from the parameters listed here, *`<AuthBy SIMMAP>`* inherits other parameters from *`<AuthBy SIMSQL>`* on page 21 and *`<AuthBy SQL>`*. The *`<AuthBy SQL>`* parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section <AuthBy SQL>.

4.8.1. AuthorisedHook

This is a Perl hook, which is called when the client is authenticated and authorised, before the **`EAP Success`** message is sent. *`AuthorisedHook`* is called with 5 arguments:

- **`$_[0]`**

  This is a pointer to this *`AuthBy`*.

- **`$_[1]`**

  This is a pointer to the current EAP context structure of the current user.

- **`$_[2]`**

  This is a pointer to the last request from the client.

- **`$_[3]`**

  This is a pointer to the authentication result. It is preset to **`main::ACCEPT`** but it can be changed if needed.

- **`$_[4]`**

  This is a pointer to the authentication reason. It is preset to empty string but it can be changed if needed.

4.8.2. AutoMPPEKeys

This is a flag. When set, *`MS-MPPE-Send-Key`* and *`MS-MPPE-Recv-Key`* are automatically provided in Access-Accept in order to set dynamic WEP (Wired Equivalent Privacy) keys.

4.8.3. EAPType

This string defines the supported authentication methods.

4.8.4. MAP

This object list lists the *<MAP>* on page 26 clauses used by this *<AuthBy SIMMAP>*.

4.8.5. MaxReauthentications

This integer defines the maximum number of permitted consecutive reauthentications. The default value is **`100`**.

4.8.6. NumTriplets

This integer defines the number of triplets requested for SIM authentication. The number must match to what the client is expecting, otherwise the authentication fails. The value is either **`2`** or **`3`**, the default value is **`3`**.

4.8.7. ReauthenticationRealm

This string defines if *`ReauthenticationRealm`* is appended to the reauthentication ID. It ensures that RADIUS routing gets the reauthentication request back to the server, which handles the request. This has no default value.

The following formatter is available in *`ReauthenticationRealm`*:

- *`%0`*

  This is the realm part of identity.

For more information about formatters, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section Special characters.

4.8.8. RequireVersion

This flag defines if the SIM client version is required or not. The recent versions of the EAP-SIM standard require the SIM client to send a protocol version information with each authentication. Some clients (for example, Cisco V5) do not send the protocol version. The default value is **`1`**. Set this parameter to **`0`** only when you must support clients that do not send the version number.

4.8.9. SupportVersions

This string controls which versions of EAP-SIM standard are supported. The parameter is a list of comma-separated version numbers, listed in order of preference, the most preferred first. The default value is **`0,1`** which means that all versions are supported.

4.8.10. UseResultInd

This flag enables the *`AT_RESULT_IND`* usage. *`AT_RESULT_IND`* is a flag which, when set, requires result indications if they are also enabled in the peer. This is not set by default.

4.8.11. UseTMSI

This flag defines whether the server offers and accepts pseudonyms. This is not set by default.

4.8.12. GetTMSIQuery

This string contains the SQL query for mapping the IMSI from a TMSI.

The following bind variable is available in *`GetTMSIQuery`*:

- *`%0`*

  This is TMSI.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *`GetTMSIQuery`*

In the following example query, bind variables marked with question marks are used with *`GetTMSIQueryParam`* listed below the query.

```
GetTMSIQuery select IMSI from SIMTMSI where TMSI = ?
GetTMSIQueryParam %0
```

Instead of the previous example, you can also use the following query without the *`GetTMSIQueryParam`*. In this case, SQL quoted value is used to create the SQL query.

```
GetTMSIQuery select IMSI from SIMTMSI where TMSI = %0
```

4.8.13. SaveTMSIQuery

This string contains the SQL query for saving the IMSI-TMSI mapping.

The following bind variables are available in *`SaveTMSIQuery`*:

- *`%0`*

  This is IMSI.

- *`%1`*

  This is TMSI.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *`SaveTMSIQuery`*

In the following example query, bind variables marked with question marks are used with *`SaveTMSIQueryParam`*s listed below the query.

```
SaveTMSIQuery replace SIMTMSI (IMSI, TMSI) values (?, ?)
SaveTMSIQueryParam %0
SaveTMSIQueryParam %1
```

Instead of the previous example, you can also use the following query without the *`SaveTMSIQueryParam`*. In this case, SQL quoted values are used to create the SQL query.

```
SaveTMSIQuery replace SIMTMSI (IMSI, TMSI) values (%0, %1)
```

4.8.14. UseReauthentication

This flag defines if the server offers and accepts the reauthentication IDs. This is not set by default.

4.8.15. DeleteReauthQuery

This string contains the SQL query for deleting the reauthentication data from the database.

The following bind variable is available in *DeleteReauthQuery*:

- *%0*

  This is the reauthentication ID.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *DeleteReauthQuery* with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*

This example query shows how *DeleteReauthQuery* is used with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*. In this example, bind variables marked with question marks are used with *DeleteReauthQueryParam* listed below the query.

```
DeleteReauthQuery update SIMUSER set  REAUTH_ID=NULL, \
      COUNTER=NULL, NONCE_S=NULL, NEXT_REAUTH_ID=NULL \
      where REAUTH_ID=?
DeleteReauthQueryParam %0
```

Instead of the previous example, you can also use the following query without the *DeleteReauthQueryParam*. In this case, SQL quoted value is used to create the SQL query.

```
DeleteReauthQuery update SIMUSER set REAUTH_ID=NULL, \
      COUNTER=NULL, NONCE_S=NULL, NEXT_REAUTH_ID=NULL \
      where REAUTH_ID=%0
```

4.8.16. GetReauthQuery

This string contains the SQL query for getting the reauthentication data from the database.

The following bind variable is available in *GetReauthQuery*:

- *%0*

  This is TMSI.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *GetReauthQuery* with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*

This example query shows how *GetReauthQuery* is used with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*. In this example, bind variables marked with question marks are used with *GetReauthQueryParam* listed below the query.

```
GetReauthQuery select IMSI, REAUTH_ID, NONCE_S, COUNTER, \
      MK, K_AUT, K_ENCR, NEXT_REAUTH_ID, VERSION from \
      SIMUSER where REAUTH_ID = ?
GetReauthQueryParam %0
```

Instead of the previous example, you can also use the following query without the *GetReauthQueryParam*. In this case, SQL quoted value is used to create the SQL query.

```
GetReauthQuery select IMSI, REAUTH_ID, NONCE_S, COUNTER, \
      MK, K_AUT, K_ENCR, NEXT_REAUTH_ID, VERSION from \
      SIMUSER where REAUTH_ID = %0
```

4.8.17. SaveReauthQuery

This string contains the SQL query for saving the reauthentication data to the database.

The following bind variables are available in *SaveReauthQuery*:

- *%0*

  This is the reauthentication ID.

- *%1*

  This is the IMSI.

- *%2*

  This is the counter.

- *%3*

  This is the master key.

- *%4*

  This is the authentication key.

- *%5*

  This is the encryption key.

- *%6*

  This is the version of SIM.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *SaveReauthQuery* with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*

This example query shows how *SaveReauthQuery* is used with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*. Bind variables marked with question marks are used with *SaveReauthQueryParam*s listed below the query.

```
SaveReauthQuery replace SIMUSER (IMSI, REAUTH_ID, COUNTER, \
      MK, K_AUT, K_ENCR, VERSION) values (?, ?, ?, ?, ?, ?, ?)
SaveReauthQueryParam %1
SaveReauthQueryParam %0
SaveReauthQueryParam %2
SaveReauthQueryParam %3
SaveReauthQueryParam %4
SaveReauthQueryParam %5
SaveReauthQueryParam %6
```

Instead of the previous example, you can also use the following query without the *SaveReauthQueryParam*s. In this case, SQL quoted values are used to create the SQL query.

```
SaveReauthQuery replace SIMUSER (IMSI, REAUTH_ID, COUNTER, \
      MK, K_AUT, K_ENCR, VERSION) values (%1, %0, %2, %3, %4, %5, %6)
```

4.8.18. UpdateReauthQuery

This string contains the SQL query for updating the reauthentication data to the database.

The following bind variables are available in *UpdateReauthQuery*:

- *%0*

  This is the reauthentication ID.

- *%1*

  This is the IMSI.

- *%2*

  This is the counter.

- *%3*

  This is *NONCE_S*, the nonce value from the *AT_NONCE_S* attribute.

- *%4*

  This is the ID of the next reauthentication.

- *%5*

  This is the EAP user name.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *UpdateReauthQuery*

In the following example query, bind variables marked with question marks are used with *UpdateReauthQueryParam* listed below the query.

```
UpdateReauthQuery update SIMUSER set  REAUTH_ID=?, COUNTER=?, \
      NONCE_S=?, NEXT_REAUTH_ID=? where IMSI=?
UpdateReauthQuery %0
UpdateReauthQuery %2
UpdateReauthQuery %3
UpdateReauthQuery %4
UpdateReauthQuery %1
```

Instead of the previous example, you can also use the following query without the *UpdateReauthQueryParam*. In this case, SQL quoted values are used to create the SQL query.

```
UpdateReauthQuery update SIMUSER set  REAUTH_ID=%0, COUNTER=%2, \
      NONCE_S=%3, NEXT_REAUTH_ID=%4 where IMSI=%1
```

4.9. <AuthBy SIMREST>

This section describes the configuration parameters of *<AuthBy SIMREST>*. Apart from the parameters listed here, *<AuthBy SIMREST>* inherits other parameters from *<AuthBy SIMSQL>* on page 21.

REST interface is provided by Radiator's HTTP client module. HTTP client parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section *`HTTP client configuration`*.

`<AuthBy SIMREST>` fetches authentication vectors over a HTTP or HTTPS REST interface. See `goodies/eap_sim_rest.cfg` for a sample configuration. The REST API is documented in `goodies/rest.txt`.

4.9.1. NoReplyReject

When set, this optional flag parameter causes Radiator to REJECT the request instead of returning IGNORE if badly formatted REST reply or no RESTreply at all is received. This parameter is not set by default.

4.10. <AuthBy SIMSQL>

This section describes the configuring parameters of *`<AuthBy SIMSQL>`*. Apart from the parameters listed here, *`<AuthBy SIMSQL>`* inherits other parameters from *`<AuthBy SIM>`* on page 14 and Radiator SQL database module. The inherited SQL database parameters are:

- *`ConnectionAttemptFailedHook`*
- *`ConnectionHook`*
- *`DateFormat`*
- *`DBAuth`*
- *`DBSource`*
- *`DBUsername`*
- *`DisconnectAfterQuery`*
- *`FailureBackoffTime`*
- *`NoConnectionsHook`*
- *`RoundRobinOnFailure`*
- *`SQLRecoveryFile`*
- *`SQLRetries`*
- *`Timeout`*

These parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section <AuthBy SQL>.

4.10.1. GetTMSIQuery

This string contains the SQL query for mapping the IMSI from a TMSI.

The following bind variable is available in *`GetTMSIQuery`*:

- *`%0`*

  This is TMSI.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *`GetTMSIQuery`*

In the following example query, bind variables marked with question marks are used with *`GetTMSIQueryParam`* listed below the query.

```
GetTMSIQuery select IMSI from SIMTMSI where TMSI = ?
GetTMSIQueryParam %0
```

Instead of the previous example, you can also use the following query without the *GetTMSIQueryParam*. In this case, SQL quoted value is used to create the SQL query.

```
GetTMSIQuery select IMSI from SIMTMSI where TMSI = %0
```

4.10.2. GetTMSIQueryParam

This string array defines the bind variables to be used with *GetTMSIQuery*. See *GetTMSIQuery* on page 10 for more information about the available bind variables.

4.10.3. SaveTMSIQuery

This string contains the SQL query for saving the IMSI-TMSI mapping.

The following bind variables are available in *SaveTMSIQuery*:

- *%0*

  This is IMSI.

- *%1*

  This is TMSI.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *SaveTMSIQuery*

In the following example query, bind variables marked with question marks are used with *SaveTMSIQueryParam*s listed below the query.

```
SaveTMSIQuery replace SIMTMSI (IMSI, TMSI) values (?, ?)
SaveTMSIQueryParam %0
SaveTMSIQueryParam %1
```

Instead of the previous example, you can also use the following query without the *SaveTMSIQueryParam*. In this case, SQL quoted values are used to create the SQL query.

```
SaveTMSIQuery replace SIMTMSI (IMSI, TMSI) values (%0, %1)
```

4.10.4. SaveTMSIQueryParam

This string array defines the bind variables to be used with *SaveTMSIQuery*. See *SaveTMSIQuery* on page 10 for more information about the available bind variables.

4.10.5. DeleteReauthQuery

This string contains the SQL query for deleting the reauthentication data from the database.

The following bind variable is available in *DeleteReauthQuery*:

- *%0*

  This is the reauthentication ID.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *DeleteReauthQuery* with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*

This example query shows how *DeleteReauthQuery* is used with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*. In this example, bind variables marked with question marks are used with *DeleteReauthQueryParam* listed below the query.

```
DeleteReauthQuery update SIMUSER set  REAUTH_ID=NULL, \
      COUNTER=NULL, NONCE_S=NULL, NEXT_REAUTH_ID=NULL \
      where REAUTH_ID=?
DeleteReauthQueryParam %0
```

Instead of the previous example, you can also use the following query without the *DeleteReauthQueryParam*. In this case, SQL quoted value is used to create the SQL query.

```
DeleteReauthQuery update SIMUSER set REAUTH_ID=NULL, \
      COUNTER=NULL, NONCE_S=NULL, NEXT_REAUTH_ID=NULL \
      where REAUTH_ID=%0
```

4.10.6. DeleteReauthQueryParam

This string array defines the bind variables to be used with *DeleteReauthQuery*. See *DeleteReauthQuery* on page 11 for more information about the available bind variables.

4.10.7. GetReauthQuery

This string contains the SQL query for getting the reauthentication data from the database.

The following bind variable is available in *GetReauthQuery*:

- *%0*

  This is TMSI.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *GetReauthQuery* with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*

This example query shows how *GetReauthQuery* is used with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*. In this example, bind variables marked with question marks are used with *GetReauthQueryParam* listed below the query.

```
GetReauthQuery select IMSI, REAUTH_ID, NONCE_S, COUNTER, \
      MK, K_AUT, K_ENCR, NEXT_REAUTH_ID, VERSION from \
      SIMUSER where REAUTH_ID = ?
GetReauthQueryParam %0
```

Instead of the previous example, you can also use the following query without the *GetReauthQueryParam*. In this case, SQL quoted value is used to create the SQL query.

```
GetReauthQuery select IMSI, REAUTH_ID, NONCE_S, COUNTER, \
      MK, K_AUT, K_ENCR, NEXT_REAUTH_ID, VERSION from \
      SIMUSER where REAUTH_ID = %0
```

4.10.8. GetReauthQueryParam

This string array defines the bind variables to be used with *GetReauthQuery*. See *GetReauthQuery* on page 11 for more information about the available bind variables.

4.10.9. SaveReauthQuery

This string contains the SQL query for saving the reauthentication data to the database.

The following bind variables are available in *SaveReauthQuery*:

- *%0*

  This is the reauthentication ID.

- *%1*

  This is the IMSI.

- *%2*

  This is the counter.

- *%3*

  This is the master key.

- *%4*

  This is the authentication key.

- *%5*

  This is the encryption key.

- *%6*

  This is the version of SIM.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *SaveReauthQuery* with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*

This example query shows how *SaveReauthQuery* is used with *<AuthBy SIMSQL>* and *<AuthBy SIMMAP>*. Bind variables marked with question marks are used with *SaveReauthQueryParam*s listed below the query.

```
SaveReauthQuery replace SIMUSER (IMSI, REAUTH_ID, COUNTER, \
      MK, K_AUT, K_ENCR, VERSION) values (?, ?, ?, ?, ?, ?, ?)
SaveReauthQueryParam %1
SaveReauthQueryParam %0
SaveReauthQueryParam %2
SaveReauthQueryParam %3
SaveReauthQueryParam %4
SaveReauthQueryParam %5
SaveReauthQueryParam %6
```

Instead of the previous example, you can also use the following query without the *SaveReauthQueryParam*s. In this case, SQL quoted values are used to create the SQL query.

```
SaveReauthQuery replace SIMUSER (IMSI, REAUTH_ID, COUNTER, \
      MK, K_AUT, K_ENCR, VERSION) values (%1, %0, %2, %3, %4, %5, %6)
```

4.10.10. SaveReauthQueryParam

This string array defines the bind variables to be used with *SaveReauthQuery*. See *SaveReauthQuery* on page 19 for more information about the available bind variables.

4.10.11. UpdateReauthQuery

This string contains the SQL query for updating the reauthentication data to the database.

The following bind variables are available in *UpdateReauthQuery*:

- *%0*

  This is the reauthentication ID.

- *%1*

  This is the IMSI.

- *%2*

  This is the counter.

- *%3*

  This is *NONCE_S*, the nonce value from the *AT_NONCE_S* attribute.

- *%4*

  This is the ID of the next reauthentication.

- *%5*

  This is the EAP user name.

For more information about SQL bind variables, see Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section SQL Bind Variables.

Example: *UpdateReauthQuery*

In the following example query, bind variables marked with question marks are used with *UpdateReauthQueryParam* listed below the query.

```
UpdateReauthQuery update SIMUSER set  REAUTH_ID=?, COUNTER=?, \
      NONCE_S=?, NEXT_REAUTH_ID=? where IMSI=?
UpdateReauthQuery %0
UpdateReauthQuery %2
UpdateReauthQuery %3
UpdateReauthQuery %4
UpdateReauthQuery %1
```

Instead of the previous example, you can also use the following query without the *UpdateReauthQueryParam*. In this case, SQL quoted values are used to create the SQL query.

```
UpdateReauthQuery update SIMUSER set  REAUTH_ID=%0, COUNTER=%2, \
      NONCE_S=%3, NEXT_REAUTH_ID=%4 where IMSI=%1
```

4.10.12. UpdateReauthQueryParam

This string which defines the bind variables to be used with *UpdateReauthQuery*. See *UpdateReauthQuery* on page 20 for more information about the available bind variables.

4.11. <AuthBy SIMWX>

This section describes the configuring parameters of *<AuthBy SIMWX>*. Apart from the parameters listed here, *<AuthBy SIMWX>* inherits other parameters from *<AuthBy SIMSQL>* on page 21 and *<WxClient>* on page 34.

4.11.1. ConvertAKAVectors

This flag defines whether the AKA vectors received via Diameter Wx or SWx are converted to SIM triplets. Enable this option if the HSS cannot return 2G triplets over Wx interface. This is not set by default but if *Interface* is set to **SWx**, this is enabled automatically.

4.12. <MAP>

This section describes the configuration parameters of *<MAP>*. Apart from the parameters listed here, all parameters from *<PeerSP>* on page 26 are available for setting the default values for *<PeerSP>* clauses within a *<MAP>*.

4.12.1. NumVectors

This integer defines the number of authentication vectors requested via MAP, if the number of the vectors is not specified in the MAP request. This parameter is used with version 2 *SendAuthenticationInfo* requests which do not specify the number of requested vectors. The default value is **3**.

4.13. <ServerSIGTRAN> within <MAP>

<ServerSIGTRAN> does not have configurable parameters at the moment. It inherits parameters from common *StreamServer* class. These parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section <ServerDiameter>.

A *<ServerSIGTRAN>* is required if connections need to be accepted from peer IPSPs or ASPs.

4.14. <PeerSP xxxxxx> within <MAP>

This section describes the configuration parameters of *<PeerSP>*. A *<PeerSP>* defines parameters and settings for a SIGTRAN IPSP or ASP a *<MAP>* uses.

To be able to use *<PeerSP>*, you must have `Convert::ASN1` Perl module version 0.26 or later.

4.14.1. AckTimeout

This integer defines the timeout in seconds for how long to wait M3UA ASP (Application Server Process) Up and ASP Active responses. This is also used for scheduling peering restart with ASP Up when an error is received because peering is not up. The default value is **2**.

4.14.2. ACVersionForSAI

This integer defines the application context version for *sendAuthenticationInfo* operation. The value can be **2** (3GPP TS 09.02) or **3** (3GPP TS 29.002). The default value is **3**. When AKA quintuplets are set, this parameter must be **3**.

4.14.3. ASPIdentifier

This integer defines the optional value for *ASP Identifier* parameter in M3UA messages.

4.14.4. DebugASN

This flag enables `ASN.1` library debug dumps to be printed to *`STDERR`*.

4.14.5. DestPointCode

This integer defines the destination point code in M3UA payload data messages. This is a mandatory parameter and must be defined as it has no default value.

4.14.6. HeartbeatTimeout

This optional parameter defines the timeout in seconds that is used when waiting for M3UA Heartbeat responses. Setting this to a non-zero value enables peer availability checks with M3UA Heartbeat messages. This is not set by default.

If there are no M3UA messages received after 2 consecutive Heartbeat messages, the peer is automatically disconnected.

```
# Monitor connectivity to the peer with M3UA heartbeat messages
HeartbeatTimeout 3
```

Note

Heartbeat messages are always responded to. You do not have to set *`HeartbeatTimeout`* for responses.

4.14.7. HeartbeatData

This parameter defines an optional value that is sent with M3UA Heartbeat messages. Special formatting characters are allowed. This is not set by default.

Here is an example of using *`HeartbeatData`*:

```
HeartbeatData Radiator SIGTRAN stack heartbeat %t
```

4.14.8. ImmediateResponsePreferredInSAI

This flag defines whether *`immediateResponsePreferred`* is sent with *`sendAuthenticationInfoArg`*. This is set by default.

4.14.9. InfoString

This string defines the value for *`INFO String`* parameter in M3UA messages. This is an optional parameter, it has no default value and thus no *`INFO String TLV`* is added by default.

4.14.10. Initiator

This flag opens the connection to the peer, when set.

4.14.11. LocalAddress and LocalPort

These parameters control the address and optionally the port number used for the client source port, although this is usually not necessary. *`LocalPort`* is a string, it can be a port number or name. It binds the local port if *`LocalAddress`* is defined. If *`LocalPort`* is not specified or if it is set to **`0`**, a port number is allocated in the usual way.

When SCTP multihoming is supported, multiple comma separated addresses can be configured. All addresses defined with *LocalAddress* must be either IPv4 or IPv6 addresses.

```
LocalAddress 203.63.154.29
LocalPort 12345
```

4.14.12. NetworkAppearance

This integer defines the network appearance in M3UA payload data messages. This has no default value and thus no *Network Appearance TLV* is added.

4.14.13. NetworkIndicator

This string defines the network indicator name or number in M3UA payload data messages. The default value is **National**. You can define the value with either as a numerical value or as text. The allowed values are shown in the table below.

Table 1. Available values of NetworkIndicator

Numerical value	Text
0	**International**
1	**Spare international**
2	**National**
3	**Spare national**

4.14.14. OrigPointCode

This integer defines the originating point code in M3UA payload data messages. This is a mandatory parameter and must be defined as it has no default value.

4.14.15. OurGlobalTitle

This string defines the global title number in calling party address when it is sent to the peer. This has no default value.

4.14.16. OurGlobalTitleIndicator

This integer defines the value of global title indicator in calling party address when it is sent to the peer. The value is either **0** or **4**, the default value is **4** (the global title includes translation type, numbering plan, encoding scheme and nature of address indicator).

4.14.17. OurNatureOfAddress

This string defines the nature of address name or number in calling party address when it is sent to the peer. The default value is **international**. This parameter has the same allowed values as *PeerNatureOfAddress*. For a complete list of available parameters, see Table 2. Available values of PeerNatureOfAddress on page 29.

4.14.18. OurNumberingPlan

This string defines the numbering plan name or number in calling party address when it is sent to the peer. The default value is **ISDN/telephony**. This parameter has the same allowed values as *PeerNumberingPlan*. For a complete list of available parameters, see Table 3. Available values of PeerNumberingPlan on page 30.

4.14.19. OurRoutingIndicator

This string defines the routing indicator name or number in calling party address when it is sent to peer. The default value is **`GT (Route on GT)`**. This parameter has the same allowed values as *`PeerRoutingIndicator`*. For a complete list of available parameters, see Table 4. Available values of PeerRoutingIndicator on page 30.

4.14.20. OurSubSystemNumber

This string defines the subsystem number or name in calling party address when it is sent to the peer. The default value is **`149`** (SGSN (Serving GPRS Support Node)). This parameter has the same allowed values as *`PeerSubSystemNumber`*. For a complete list of available parameters, see Table 5. Available values of PeerSubSystemNumber on page 30.

4.14.21. OurTranslationType

This string defines the translation type number or name in calling party address when it is sent to the peer. The default value is **`0`**.

4.14.22. PeerGlobalTitle

This string defines the global title number in called party address when it is sent to the peer. This has no default value.

4.14.23. PeerGlobalTitleHook

This is a Perl hook, which creates peer global title from the IMSI and returns the global title. If no value is returned, *`PeerGlobalTitle`* is used as the global title. This has no default value. *`PeerGlobalTitleHook`* is called with 1 argument:

- **`$_[0]`**

  This is the IMSI.

See `goodies/sigtran-peer-gt-hook.pl` for a sample *`PeerGlobalTitleHook.`*

```
PeerGlobalTitleHook file:"%D/sigtran-peer-gt-hook.pl"
```

4.14.24. PeerGlobalTitleIndicator

This integer defines the value of global title indicator in called party address when it is sent to the peer. The value can be **`0`** or **`4`**, the default value is **`4`** (the global title includes translation type, numbering plan, encoding scheme and nature of address indicator).

4.14.25. PeerNatureOfAddress

This string defines the nature of address name or number in called party address when it is sent to the peer. The default value is **`international`**. You can define the value with either as a numerical value or as text. The allowed values are shown in the table below.

Table 2. Available values of PeerNatureOfAddress

Numerical value	Text
`0`	**`unknown`**
`1`	**`subscriber`**
`2`	**`reserved for national use`**

Numerical value	Text
3	national significant
4	international

4.14.26. PeerNumberingPlan

This string defines the numbering plan name or number in called party address when it is sent to the peer. The default value is **ISDN/mobile**. You can define the value with either as a numerical value or as text. The allowed values are shown in the table below.

Table 3. Available values of PeerNumberingPlan

Numerical value	Text
0	unknown
1	ISDN/telephony
2	generic
3	data
4	telex
5	maritime mobile
6	land mobile
7	ISDN/mobile
14	private network or network-specific
15	reserved

4.14.27. PeerRoutingIndicator

This string defines the routing indicator name or number in called party address when it is sent to peer. The default value is **Route on GT**. You can define the value with either as a numerical value or as text. The allowed values are shown in the table below.

Table 4. Available values of PeerRoutingIndicator

Numerical value	Text
0	Route on GT
1	Route on SSN

4.14.28. PeerSubSystemNumber

This string defines the subsystem number or name in called party address when it is sent to the peer. The default value is **149** (**SGSN**). You can define the value with either as a numerical value or as text. The allowed values are shown in the table below.

Table 5. Available values of PeerSubSystemNumber

Numerical value	Text
0	Not used/Unknown

Numerical value	Text
1	**SCCP management**
3	**ISDN user part**
4	**OMAP**
5	**MAP**
6	**HLR**
7	**VLR**
8	**MSC**
9	**EIC**
10	**AUC**
11	**ISDN supplementary services**
142	**RANAP**
143	**RNSAP**
145	**GMLC**
146	**CAP**
147	**gsmSCF or IM-SSF**
148	**SIWF**
149	**SGSN**
150	**GGSN**
241	**INAP**
249	**PCAP**
250	**BSC**
251	**MSC**
252	**SMLC**
253	**BSS O&M**
254	**A interface**

4.14.29. PeerTranslationType

This string defines the translation type number or name in called party address when it is sent to the peer. The default value is **0**.

4.14.30. Port

This string defines the peer port the *<PeerSP>* connects to.

4.14.31. RequestingNodeTypeInSAI

This string defines the *RequestingNodeType* number in *sendAuthenticationInfoArg*. The value can be **0** (VLR (Visitor Location Register)) or **1** (SGSN). This has no default value and *RequestingNodeType* is not sent with *sendAuthenticationInfoArg*.

4.14.32. RoutingContext

This split string array defines a comma-separated list of routing context values we advertise to peer service provider. This has no default value.

4.14.33. SCTPPeer

This parameter specifies one host name or address of an SCTP peer to connect to. An address can be an IPv4 or IPv6 address. Multiple *SCTPPeer* parameters are supported. When *SCTPPeer* is defined, it is used instead of *Host* or *Peer* parameters. Special formatting characters are supported. If SCTP multihoming is not supported, connection is attempted to each peer at a time.

When SCTP multihoming is supported, connection is attempted to all peers at once. In this case, all addresses defined with *SCTPPeer* must be either IPv4 or IPv6 addresses

Here is an example of using *SCTPPeer*:

```
# Peer has multiple IPv6 addresses
SCTPPeer 2001:db8:1500:1::a100
SCTPPeer 2001:db8:1500:2::a100
```

4.14.34. SEModeSendASPUP

This flag defines if Radiator sends *ASPUP* when running in Single Exchange IPSP (IP Server Process) mode. This is not set by default.

4.14.35. SEModeWaitNotify

This flag defines if Radiator waits for *NTFY(AS-INACTIVE)* after *ASP Up Ack* and *NTFY(AS-ACTIVE)* before sending traffic to the peer. When set, notify messages are needed before traffic is sent to the peer which is typical for SGP-ASP communication where Radiator acts as an ASP. Otherwise *ASP Active* is sent immediately after *ASP Up Ack* which is typical for IPSP-IPSP communication. This is not set by default which means *ASP Active* is sent immediately after *ASP UP Ack*.

4.14.36. SPGroup

SPGroup is a string parameter for defining a group from which the reply for a MAP request is allowed from. *SPGroup* is not set by default and reply for a request must come from the SP that the request was originally sent to.

Here is a brief example of using *SPGroup*:

```
<MAP>
    # Set the default group for all SPs
    SPGroup group1

    <PeerSP 10.2.3.4>
        # This SP belongs to its own group
        SPGroup group2
    </PeerSP>
```

```
    <PeerSP 10.2.3.5>
        # SPGroup is not set, use value from the enclosing <MAP>
    </PeerSP>
    <PeerSP 10.2.3.6>
        # SPGroup is not set, use value from the enclosing <MAP>
    </PeerSP>
</MAP>
```

4.14.37. Timeout

This integer defines the used timeout when waiting for MAP responses.

4.14.38. TriggerFailure

When *`TriggerFailure`* is set, timed out requests, TCAP Abort primitive, MAP errors, and broken messages trigger a failure indication to upper layers, such as EAP-AKA, when the received message can be mapped to a sent message. Enabling *`TriggerFailure`* allows Radiator to reject authentication attempts instead of ignoring them. *`TriggerFailure`* is not set by default.

By default errors and time outs are logged and otherwise ignored. When the client has more than one RADIUS server configured, it can switch to a secondary server if the current server does not answer. This helps recovering from transient connectivity and other problems.

Here is an example of using *`TriggerFailure`*:

```
# We don't want cause time outs to RADIUS client
TriggerFailure
```

4.15. <ServerWXMAP>

This section describes the configuring parameters of *<ServerWXMAP>*. Apart from the parameters listed here, *<ServerWXMAP>* inherits other parameters from *`AuthGeneric`*. These parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section <ServerDIAMETER>.

4.15.1. 3GPPCardDatabaseFilename

This string defines the file path and name where the 2G SIM or 3G USIM card details are stored. Radiator requires read and write access to this file. When defined, this parameter is used to find the Milenage algorithm parameters for SIM and USIM cards. See `goodies/simcards.dat` for a sample file.

The file contains the following information coded to hexadecimal:

- IMSI for the SIM/USIM card
- Ki
- Encrypted OPc
- AMF
- SQN

4.15.2. IndLength

This integer defines the length of the IND part in bits in the AKA authentication vector SQN. The default value is **5**.

4.15.3. TripletsFile

This string defines the path and name of the file containing the pre-generated triplets. When defined, Radiator extracts the triplets from the specified file. If extracting fails or the file path and name are not defined, Radiator extracts the Milenage algorithms from a file specified in *3GPPCardDatabaseFilename* on page 13 and computes the triplets from the Milenage data.

4.16. <WxClient>

This section describes the configuring parameters of *<WxClient>*. Apart from the parameters listed here, *<ServerWXMAP>* inherits other parameters from DiaClient except *PostDiaToRadiusConversionHook* and *PostRadiusToDiaConversionHook*. These parameters are documented in Radiator reference manual [https://files.radiatorsoftware.com/radiator/ref.pdf] under section <AuthBy DIAMETER>.

4.16.1. Interface

This string defines the interface supported by the *<WxClient>*. Possible values are **Wx** and **SWx**.

4.16.2. LocalAddress and LocalPort

These parameters control the address and optionally the port number used for the client source port, although this is usually not necessary. *LocalPort* is a string, it can be a port number or name. It binds the local port if *LocalAddress* is defined. If *LocalPort* is not specified or if it is set to **0**, a port number is allocated in the usual way.

When SCTP multihoming is supported, multiple comma separated addresses can be configured. All addresses defined with *LocalAddress* must be either IPv4 or IPv6 addresses.

```
LocalAddress 203.63.154.29
LocalPort 12345
```

4.17. <IMSICrypt>

This section describes the configuration parameters of an *<IMSICrypt>* clause. This clause provides support for Permanent Identity encryption, sometimes also called IMSI encryption or IMSI privacy. IMSI encryption is specified in 3GPP document *S3-170116* and Wireless Broadband Alliance technical specification *IMSI Privacy Protection for Wi-Fi*.

IMSI encryption is supported by all EAP-SIM, EAP-AKA, EAP-AKA' and 3GPP AAA Server configuration clauses. To enable IMSI encryption, you need to modify Radiator configuration as follows:

- First define an *<IMSICrypt>* clause with an *Identifier* parameter.
- Then add *IMSICrypt* configuration parameter in *AuthBy* clauses.

For required software versions and modules, see Section 2.1. Prerequisites on page 2. A full configuration example is in file `goodies/imsicrypt.cfg`

Example: *IMSICrypt*

For a full example, see `goodies/simcrypt.cfg`. Key configured with *DefaultPrivateKey* is used when **Key Identifier AVP**, also known as **certificate identifier attribute**, is not present. Key configured with *PrivateKeyFile* is used identifier is not present.

For more information about **Key Identifier AVP** and **certificate identifier attribute**, see 3GPP document *S3-170116* and Wireless Broadband Alliance technical specification *IMSI Privacy Protection for Wi-Fi*.

```
<IMSICrypt>
   # Identifier is used by AKA and SIM clauses to refer to this
   # clause for identity decryption.
   Identifier imsi-decrypter

   # DefaultPrivateKeyFile and DefaultPrivateKeyPassword work as
   # pairs.
   DefaultPrivateKeyFile %D/certificates/server-key.pem
   DefaultPrivateKeyPassword whatever

   #DefaultPrivateKeyFile %D/private-keys/default-key1.pem
   #DefaultPrivateKeyPassword password-for-default-key1

   #DefaultPrivateKeyFile %D/private-keys/default-key2.pem
   ## Key in file default-key2.pem is not password protected

   #PrivateKeyFile     CertificateSerialNumber=12345,%D/private-keys/key-12345.pem
   #PrivateKeyPassword CertificateSerialNumber=12345,password-for-key-12345

   #PrivateKeyFile     CertificateSerialNumber=23456,%D/private-keys/key-23456.pem
   ## Key in file key-23456.pem is not password proteced

   #PrivateKeyFile     CertificateSerialNumber=34567,%D/private-keys/key-34567.pem
   #PrivateKeyPassword CertificateSerialNumber,password-for-key-34567
</IMSICrypt>

<AuthBy AKAWX>
   # Other AKAWX configuration parameters
   IMSICrypt imsi-decrypter
</AuthBy>
```

4.17.2. DefaultPrivateKeyFile

`DefaultPrivateKeyFile` defines a private key file name for a key that is used when an encrypted permanent identity does not have key identifier. You can configure multiple key files to support key roll over. Decryption is attempted with all key files until the first one succeeds. If no key is able to correctly decrypt an encrypted identity, an error is returned to the client and the authentication fails.

See the configuration example on page 34 for more information.

4.17.3. DefaultPrivateKeyPassword

`DefaultPrivateKeyPassword` defines the password for decrypting a default private key defined with *`DefaultPrivateKey`*. Key encryption is optional. If a key is stored without encryption, this parameter is not needed. An encrypted key file and its respective password must be configured in pairs.

See the configuration example on page 34 for more information.

4.17.4. PrivateKeyFile

`PrivateKeyFile` defines a private key file name in **`name=value,filename`** format. This key is used when an encrypted permanent identity sent by the client has a key identifier. Decryption is attempted only with the key that matches the key identifier the client sends. If the key is not able to correctly decrypt the encrypted

identity, an error is returned to the client and the authentication fails. You should not configure more than one *PrivateKeyFile* parameter with the same **name=value** because only the latest parameter is used.

See the configuration example on page 34 for more information.

4.17.5. PrivateKeyPassword

PrivateKeyPassword defines the password for decrypting a private key defined with *PrivateKey*. The format for this parameter is **name=value,password** where **name** and **value** must match the respective values of a *PrivateKeyFile* parameter. Key encryption is optional. If a key is stored without encryption, this parameter is not needed.

See the configuration example on page 34 for more information.

5. Abbreviations

Authentication, Authorisation, Accounting

AAA (Authentication, Authorisation, Accounting)

Acronym: **AAA**

Authentication and Key Agreement

AKA (Authentication and Key Agreement)

Acronym: **AKA**

Authentication and Key Agreement Prime

AKA' (Authentication and Key Agreement Prime)

Acronym: **AKA'**

Authentication Management Field

AMF (Authentication Management Field)

Acronym: **AMF**

Application Server Process

ASP (Application Server Process)

Acronym: **ASP**

Authentication Centre

AuC (Authentication Centre)

Acronym: **AuC**

Extensible Authentication Protocol

EAP (Extensible Authentication Protocol)

Acronym: **EAP**

Extensible Authentication Protocol - Authentication and Key Agreement

EAP-AKA (Extensible Authentication Protocol - Authentication and Key Agreement)

Acronym: **EAP-AKA**

Extensible Authentication Protocol - Authentication and Key Agreement Prime

EAP-AKA' (Extensible Authentication Protocol - Authentication and Key Agreement Prime)

Acronym: **EAP-AKA'**

Extensible Authentication Protocol - Subscriber Identity Module

EAP-SIM (Extensible Authentication Protocol - Subscriber Identity Module)

Acronym: **EAP-SIM**

Home Location Register

HLR (Home Location Register)

Acronym: **HLR**

Home Subscriber Server

HSS (Home Subscriber Server)

Acronym: **HSS**

International mobile subscriber identity

IMSI (International mobile subscriber identity)

Acronym: **IMSI**

IP Server Process

IPSP (IP Server Process)

Acronym: **IPSP**

Key Derivation Function

KDF (Key Derivation Function)

Acronym: **KDF**

Authentication key

Ki (Authentication key)

Acronym: **Ki**

Local Area Network

LAN (Local Area Network)

Acronym: **LAN**

MTP Level 3 User Adaptation Layer

M3UA (MTP Level 3 User Adaptation Layer)

Acronym: **M3UA**

Mobile Application Part

MAP (Mobile Application Part)

Acronym: **MAP**

Operator Code

OPc (Operator Code)

Acronym: **OPc**

Personal Computer Smart Card

PCSC (Personal Computer Smart Card)

Acronym: **PCSC**

Stream Control Transmission Protocol

SCTP (Stream Control Transmission Protocol)

Acronym: **SCTP**

Serving GPRS Support Node

SGSN (Serving GPRS Support Node)

Acronym: **SGSN**

Subscriber Identity Module

SIM (Subscriber Identity Module)

Acronym: **SIM**

Sequence Number

SQN (Sequence Number)

Acronym: **SQN**

Signalling System No. 7

SS7 (Signalling System No. 7)

Acronym: **SS7**

Transmission Control Protocol

TCP (Transmission Control Protocol)

Acronym: **TCP**

Temporary Mobile Subscriber Identity

TMSI (Temporary Mobile Subscriber Identity)

Acronym: **TMSI**

Universal Subscriber Identity Module

USIM (Universal Subscriber Identity Module)

Acronym: **USIM**

Visitor Location Register

VLR (Visitor Location Register)

Acronym: **VLR**

Wired Equivalent Privacy

WEP (Wired Equivalent Privacy)

Acronym: **WEP**