

Mobile ID RADIUS Integration

RADIUS Interface Gateway (RIG) - Version 1.1

Table of contents

1	Introduction	3
1.1	RIG Feature List.....	4
1.1.1	Features available with latest docker application.....	4
1.1.2	Features planned	4
1.2	Service Architecture.....	5
1.3	Authentication Methods Management.....	6
2	RIG Deployment	7
2.1	Preconditions.....	8
2.1.1	General Requirements.....	8
2.1.2	NAS Requirements.....	8
2.1.3	Connectivity Requirements	8
2.2	Customer Configuration (REDIS)	9
2.3	Docker Run.....	10
2.4	Docker Compose	11
2.5	Integration Scenarios.....	12
2.5.1	VPN Authentication, MFA, Mobile Signature	13
2.5.2	VPN Authentication, MFA, SMS OTP.....	16
2.5.3	VPN Authentication, SFA, Mobile Signature.....	19
2.5.4	VPN Authentication, MFA with local LDAP, Mobile Signature	21
2.6	SMS notifications.....	23
2.7	LDAP authentication.....	24
3	The RADIUS Protocol.....	25
3.1	Remote Authentication Dial-In User Service (RADIUS) Protocol	25
3.1.1	Protocol Overview.....	25
3.1.2	Key Concepts and Terminology.....	26
3.1.3	Basic Flow Description	28
3.1.4	RADIUS Challenge and Response.....	29
3.1.5	RADIUS via Proxies	30
4	Annexes	31
4.1	Annex 1 - List of RADIUS-related RFCs	31
4.2	Annex 2 - RADIUS Testing tools	32
4.2.1	Radclient - command line tool	32
4.2.2	RADIUS test online application	32
4.2.3	NTRadPing Test Utility - Windows desktop application	32
4.2.4	Other tools	33

Mobile ID is a brand of Swisscom.
 Swisscom (Switzerland) Ltd
 Alte Tiefenastrasse 6
 CH-3050 Bern

1 Introduction

The RIG (RADIUS Interface Gateway) application is an [API Gateway](#) for Mobile ID, exposing a RADIUS interface towards the clients and using the Mobile ID API at mobileid.swisscom.com for translating the requests of the (RADIUS-) clients into requests for the Mobile-ID service.

The Mobile ID solution protects access to your company data and applications with a comprehensive end-to-end solution for a strong multi-factor authentication (MFA). Please visit <https://mobileid.ch> for further information. Do not hesitate to [contact us](#) in case of any questions.

RIG Service (RADIUS Interface Gateway)

The RIG application is the ideal solution in a setup in which an existing RADIUS-based network is taken from single factor authentication (User-Name + User-Password) or two-factor authentication (User-Name + User-Password plus a security device challenge) to a Multi-Factor Authentication, by customizing the authentication flow of a RADIUS session and introducing a new step that uses Mobile ID strong authentication means.

During the authentication via RADIUS, an extra step will require the users to confirm the access to the service on their mobile phones.

Some clients might decide to move from 1FA (one factor authentication) to 2FA: username + password and Mobile ID. Other clients might decide to stick to 2FA but replace the existing combination of username + password and security device challenge with username + password plus Mobile ID as additional MFA.

RADIUS Users are typically defined in the format `user@domain`. However, MobileID requires the phone number (MSISDN) of the user. Therefore, the phone number of the target user should be provided with one of the possibilities listed below (RIG supports all three options):

- a) RIG service connects to a customer's Directory Service (LDAP) to retrieve the user attributes such as phone number (MSISDN) or other optional attributes such as User-Language, MobileID-serialnumber or the user's preferred authentication method.
- b) RADIUS client provides the MSISDN as part of the User-Name (`4179xxxxxxx@mydomain.com`)
- c) RADIUS client provides the MSISDN via Vendor Specific Attribute (`x-MSS-MSISDN`)

1.1 RIG Feature List

1.1.1 Features available with latest docker application

- ✓ Multitenancy
- ✓ Cloud-native microservice architecture (horizontal scalability)
- ✓ Authentication: MobileID SIM, MobileID App, OTP Text SMS
- ✓ Geofencing
- ✓ LDAP lookup (`userPassword`, `mobile`, `preferredLanguage`, `midSerial`, `preferredMFA`)
- ✓ MobileID user's serialnumber validation (optional)
- ✓ Custom RADIUS Reply Messages for Access-Reject
- ✓ Custom Text SMS Notification for specific error events

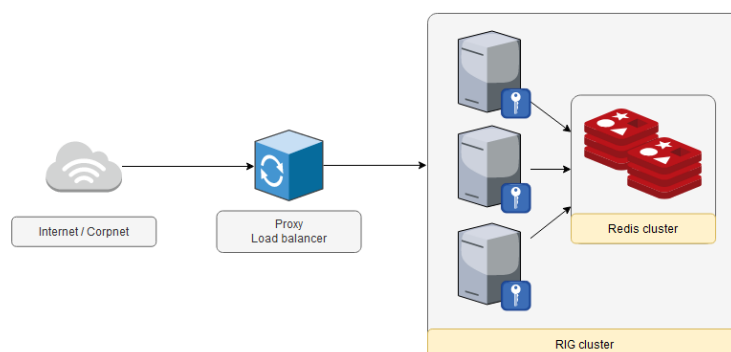
1.1.2 Features planned

- RADIUS Dynamic Fields: Allow radius clients to provide custom MSISDN, DTBD, Auth-Method
- Advanced RADIUS encryption (EAP-TLS, EAP-TTLS, PEAP, EAP-MD5, LEAP)
- RADIUS Proxy Mode (forward Access-Request after successful MFA to another 3rd party RADIUS server)

1.2 Service Architecture

The RADIUS Interface Gateway (RIG) service is developed as a cloud native microservice application. As it is the norm with this type of applications, RIG is both vertically and horizontally scalable. For latter capability, the service is designed to work well in a cluster setup, with multiple RIG nodes synchronizing between them and serving client requests. This helps with load balancing and ensuring high availability in production.

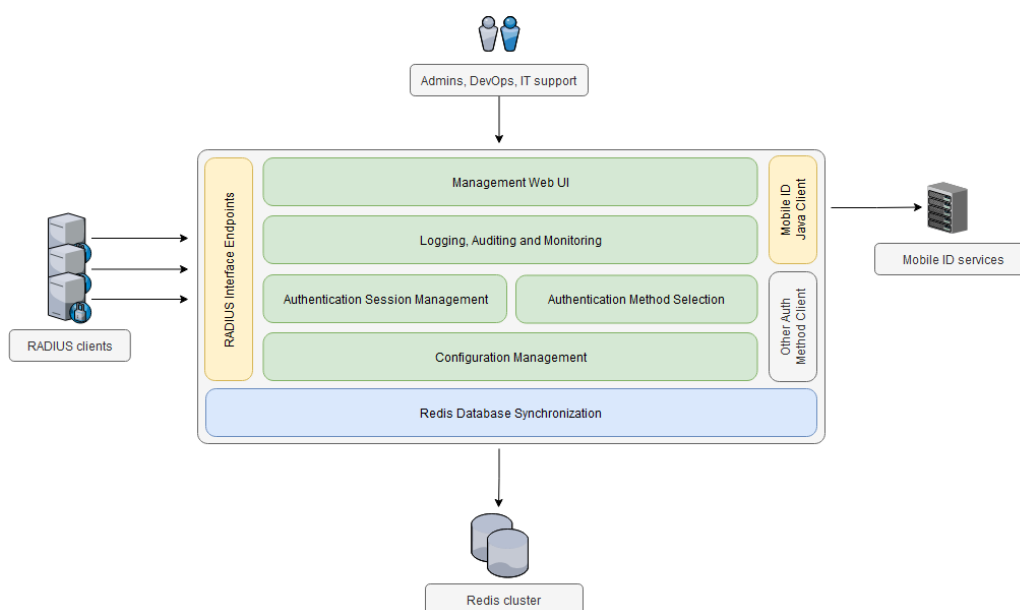
The following diagram shows a cluster setup for RIG:



Since nodes need a way of communicating between them and considering the specific target runtime environment, a good approach for this inter-node synchronization is via a common Redis database cluster. Each node will connect to this database and will be able to save and read data from all the other nodes in a RIG cluster. An ongoing authentication session, for example, will have its data stored by node A in the Redis database, with later having node B read the data for that session and carry it on.

Requests coming from outside of the RIG cluster will reach a proxy service and will then be dispatched to one appropriate RIG node. It is up to the proxy to select the right node, but any policy will work fine, as all RIG nodes have the same quality and level of data access, so any one of them could handle a request at any time. Of course, the proxy is responsible for carefully distributing the load on available nodes, but from the point of view of the RIG nodes, the functionality is the same, regardless of the chosen node.

Moving one level down in the architectural view, the following diagram shows the main logical parts of the RIG service:



1.3 Authentication Methods Management

One of the main components of the RIG service is responsible for managing the authentication methods that are available for a client and selecting the right method on a per-user basis.

The following authentication methods are currently supported:

- **SIM digital signature** - the user is prompted to confirm a transaction (RADIUS authentication session) and create a digital signature using his/her Mobile ID enabled SIM chip
- **App digital signature** - like the SIM method above, but the signing “device” is a mobile application with its keys protected by the phone system protection
- **OTP over text-SMS** - an OTP code is generated for each authentication session and sent to the user via an SMS message. A RADIUS *Access-Challenge* + *Access-Request* roundtrip is used to challenge the user to enter the OTP received via SMS.

For each RADIUS client configured for the RIG service, the following parameters need to be specified:

- The available authentication methods
- The default method, in case all the other methods are unavailable

During an authentication session, the RIG service will load this configuration and will try to match it to the current state of the user’s Mobile ID account.

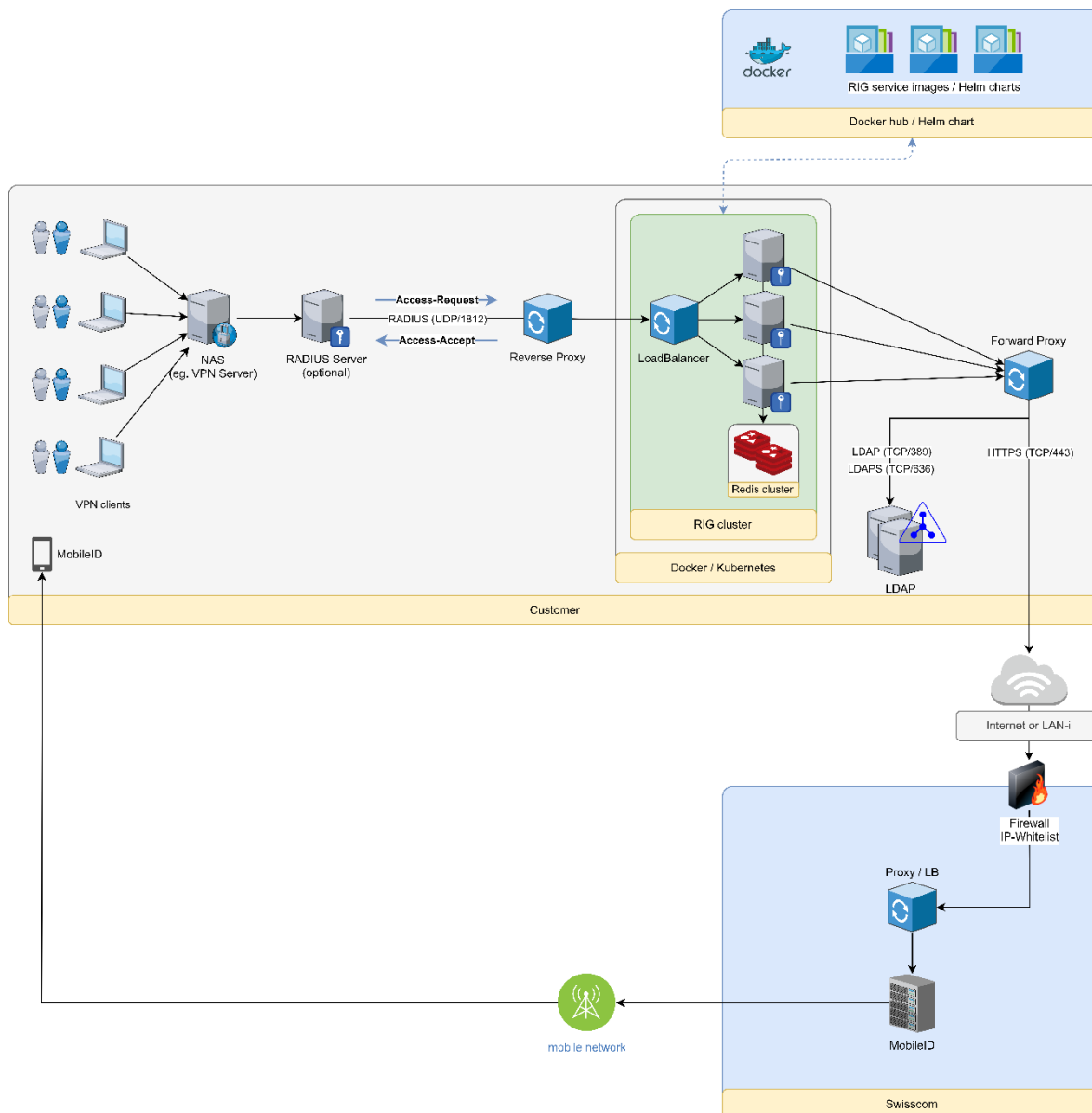
Each customer account can have its custom list of allowed MFA methods and with its own order of priority. An LDAP user attribute may be used to set a preferred MFA option per user (if the configuration parameter `UseUserMfaMethod` is set to `true`). This will always overrule any other configured order of priority.

If a preferred MFA option (from LDAP user attribute) is not active or not allowed by the customer configuration, then it will fall-back to the next MFA option according to the configured order of priority.

2 RIG Deployment

There are different types of container services you may use to deploy and run microservices like the RIG application, in a high availability setup.

We generally recommend using Kubernetes or similar container orchestration options to have a platform that automates the tasks related to the management, deployment, and scaling of container clusters.



Docker images and Helm charts may be used. The latter one facilitates the packaging and deployment of more complex setups in target Kubernetes environments. This, of course, works if the customer is already operating a Docker/Swarm environment or a Kubernetes one.

If more plain setups are needed, the RIG service and dependent components can be delivered and installed as usual software applications.

2.1 Preconditions

The following points must be considered before you proceed with the deployment.

2.1.1 General Requirements

- 1) A MobileID contract exists. You know your `AP_ID` and your API key (PFX/PKCS#12 format).
- 2) A Directory Service (LDAP) exists, which contains user attributes such as name and phone number
- 3) An appropriate security concept exists (eg. secured network communication, hardening, ...)

2.1.2 NAS Requirements

- 4) The NAS' RADIUS Access-Request packet contains either a unique `NAS-Identifier` attribute OR it has a unique and static source IP address/range. RIG will pick the configuration based on this info.
- 5) The RADIUS client `Timeout` shall be set to `60 seconds`. This will ensure enough time for the user to respond to the MobileID authentication request.
- 6) The RADIUS client `Retry` shall be set to `1`. The client should not retry because there might be still a MobileID authentication session on-going.

2.1.3 Connectivity Requirements

- 7) Connectivity is allowed as follows:
 - RADIUS (UDP/1812) requests from your NAS (eg. VPN Server) to the RIG application (or the UDP Network Load Balancer that will forward the requests to the RIG application nodes).
 - LDAP (TCP/389) or LDAPS (TCP/636) connectivity from the RIG application nodes to the LDAP host(s). RIG allows the configuration of up to three LDAP(S) host IP addresses.
 - HTTPS (TCP/443) connectivity from the RIG application nodes to the MobileID API, which is either `https://mobileid.swisscom.com:443` (if you go over the internet) or `https://195.65.233.218:443` (if you have a Swisscom LAN-I connectivity). Please note that your source IP address/range must be whitelisted in the MobileID API Firewall!
 - HTTP (TCP/80) local connectivity for the docker health check, which is optional

You may want to verify the LDAP connectivity using the command below on the node, where the RIG container application is running. You may also use ldaps (port 636) instead of ldap (port 389):

```
$ ldapwhoami -x -w <passwd> -D "cn=<tbid>,ou=<tbid>,dc=<tbid>,dc=<tbid>" -H
ldap://<ldap.myserver.com>:389
```

You may want to verify the MobileID API connectivity using the command below on the node, where the RIG container application is running:

```
$ openssl s_client -connect mobileid.swisscom.com:443
```

Note: The Firewall on the MobileID API side has a IP-based whitelist! Therefore, the source IP address/range of the request sent to the MobileID API must be in the whitelist. If the IP address is unknown, the packet will be dropped and your request will timeout.

2.2 Customer Configuration (REDIS)

Here's a [customer JSON configuration](#) example. Please add the JSON content example to your REDIS database and change the configuration according to your preferences. You may configure only one or multiple customers.

Here's a [18N Error Message JSON configuration](#) example. Please add the JSON content example to your REDIS database and change the configuration according to your preferences. This allows you to customize any Reply-Message content that are used for Access-Reject packets.

2.3 Docker Run

You can pull the Docker images from Docker Hub (or alternatively from Amazon ECR).

For a high availability you'll need these components, running **at least 2 instances** each:

- MobileID RADIUS Interface Gateway application ([Docker Hub](#) or [Amazon ECR](#))
- REDIS database using [redis](#) (requires 3 nodes for HA - alternatively [keydb](#) may be used)
- REDIS web management tool using [redis-commander](#) to manage the REDIS database content
- UDP network load balancer using [nginx](#) with this custom [nginx.conf](#) configuration

How to pull an image from Docker Hub:

```
$ docker pull mobileidch/mid-radius-rig
```

How to run a Docker application:

```
$ docker run -d -p 1812:1812/udp --env-file <my-env-file> mobileidch/mid-radius-rig
```

Here's an [env example file](#) for the RIG application. Please change the example according to your preferences.

At least the following parameters must be updated in the env file:

- `MID_CLIENT_CERTIFICATE` - This shall be your base64 encoded MobileID API key
For example: `$ base64 -w 0 <MyKey.pfx>`
- `Schnittstellen_MobileIdClient_Host` - This shall be the MobileID API endpoint, , which is either `https://mobileid.swisscom.com` (Internet) or `https://195.65.233.218` (LAN-i)

2.4 Docker Compose

With [Compose](#), we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down. With this sample, we start multiple RIG container instances.

Here's a [docker-compose.yaml example](#) with services defined as follows:

- MobileID RADIUS Interface Gateway application from [Docker Hub](#) (or alternatively from [Amazon ECR](#))
- REDIS database using [redis](#) (or alternatively [keydb](#) may be used)
- REDIS web management tool using [redis-commander](#) to manage the REDIS database content
- UDP network load balancer using [nginx](#) with this custom [nginx.conf](#) configuration

Please change the example according to your preferences.

At least the following parameters must be updated in the yaml file:

- `MID_CLIENT_CERTIFICATE` - This shall be your base64 encoded MobileID API key
For example: `$ base64 -w 0 <MyKey.pfx>`
- `Schnittstellen_MobileIdClient_Host` - This shall be the MobileID API endpoint, , which is either `https://mobileid.swisscom.com` (Internet) or `https://195.65.233.218` (LAN-i)

How to spin up using docker compose command - Please make sure that both `docker-compose.yml` and `nginx.conf` exist in the same directory before you run the command below:

```
$ docker-compose up --scale mid-radius-rig=3
```

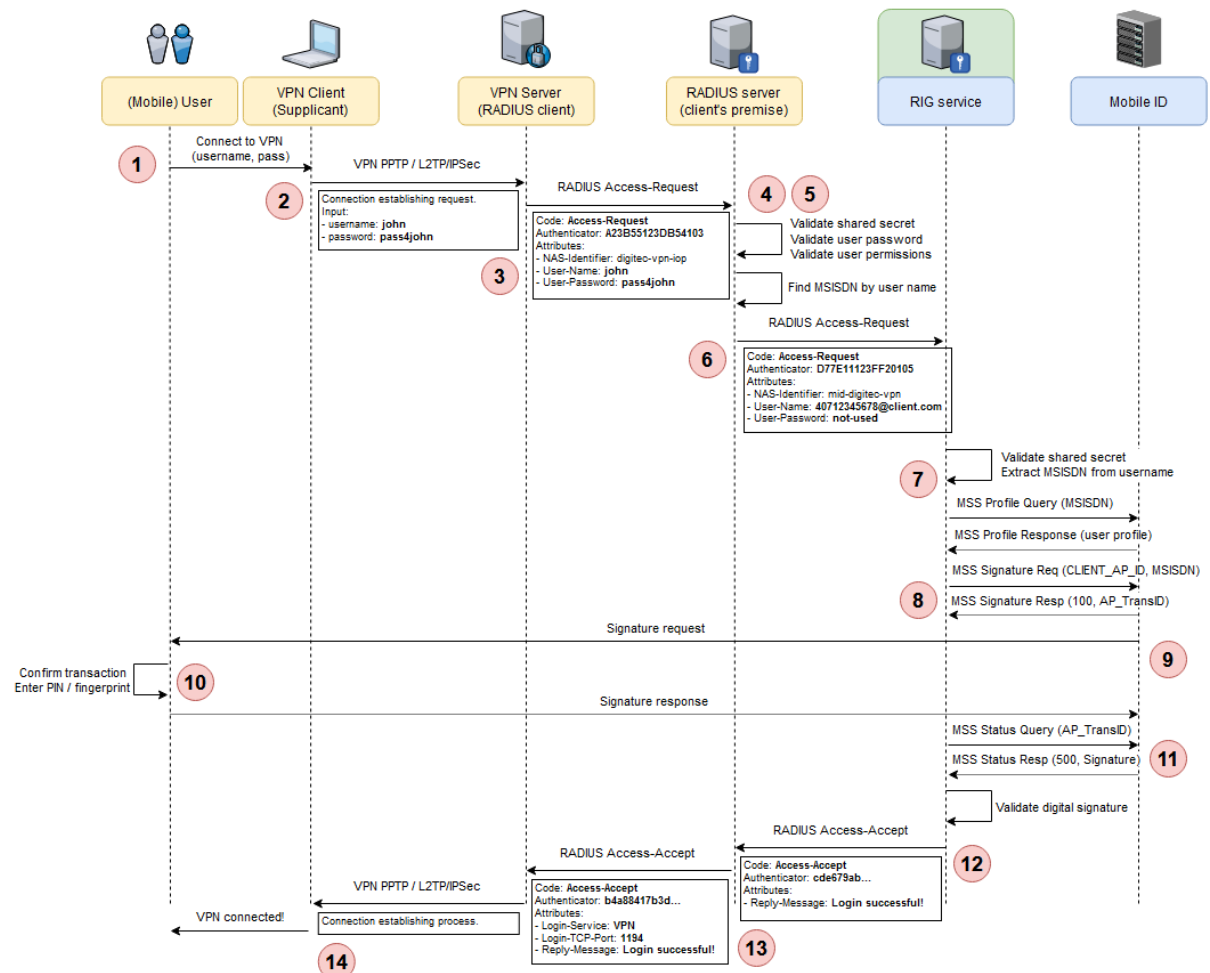
2.5 Integration Scenarios

This chapter describes four different integration scenarios.

- VPN Authentication, MFA, Mobile Signature
- VPN Authentication, MFA, SMS OTP
- VPN Authentication, SFA, Mobile Signature
- VPN Authentication, MFA with local LDAP, Mobile Signature

2.5.1 VPN Authentication, MFA, Mobile Signature

In the first scenario discussed in this document, a user wants to establish a VPN connection and the backend is configured to use an existing RADIUS server for user authentication and authorization, coupled with the RADIUS Interface Gateway (RIG) service for multi factor authentication. Moreover, the user is an active Mobile ID user, so a mobile signature is acquired in the process and, depending on the outcome of that signature validation, the user gets a new VPN connection established.



Step 1 - The user opens the VPN client (the Supplicant) on his/her laptop and clicks on the desired VPN connection. Since he/she does not want to store credentials locally, the VPN client asks him/her to enter the username and the password of the account. The user enters the requested credentials.

Step 2 - The VPN client initiates a specific protocol connection (e.g., PPTP, L2TP/IPSec, etc) to the VPN server, transmitting the username and the password on the line. The VPN server is configured to use RADIUS as authentication protocol.

Step 3 - The VPN server creates a RADIUS *Access-Request* packet using its RADIUS shared secret, the username and password received from the VPN client and the type of service that the user requested: VPN. The RADIUS packet is sent to the client's backend RADIUS server.

```
Code: Access-Request
Identifier: 15
Authenticator: D23B55123DB54103
Attributes:
- NAS-Identifier: digitec-vpn-iop
- User-Name: john
- User-Password: md5("secret"+Authenticator)
```

Step 4 - The RADIUS server validates all the aspects concerning the received request: the shared secret and the user password and permissions. It also checks if there are other extra authentication steps required for this user. It finds that this user has multi-factor authentication (MFA) enabled via Mobile ID.

Step 5 - To use Mobile ID as MFA, the flow needs to go through the RIG service. For this to happen, the RADIUS server needs the MSISDN of the currently authenticating user. It makes a local query to its data store and finds the MSISDN of the user based on the given username.

An alternative at this step is for the mobile users to directly enter the username as a construct of "<msisdn>@<domain>" (e.g., "40712345678@client.com") and directly give that value to the VPN client in Step 1 above. Moreover, as per RADIUS flow, this username is also paired to a matching password, therefore having the user enter a wrong (by typo or intentionally) MSISDN is unlikely.

Step 6 - The RADIUS server creates a second RADIUS *Access-Request* packet, like the one in **Step 3**. In this case, the RADIUS server acts as a client for the next RADIUS server in the chain: the RIG service operated by Swisscom. The packet created at this step contains the secret shared between this RADIUS server and the RIG service. The username is formatted as "<msisdn>@<domain>" (e.g., "40712345678@client.com"). The user password is not used at this step, so it is filled in with some filler text ("not-used").

```
Code: Access-Request
Identifier: 21
Authenticator: CCEB551E2254AAB
Attributes:
- NAS-Identifier: mid-digitec-vpn
- User-Name: 40712345678@client.com
- User-Password: md5("secret"+Authenticator)
```

Step 7 - The RIG service receives the request and proceeds to validate the NAS identifier and the shared secret of the client. Based on the client's configuration, the RIG service decides how to proceed and what parameters to use for the next step:

- Based on the authentication method algorithm configured for this client, an array of possible methods is selected. These can be any of SIM-based digital signature, app-based digital signature, OTP over SMS or any other method that will be implemented in future versions.
- From the **User-Name** received from the RADIUS client, the user's MSISDN is extracted
- Based on an MSS Profile Query for the user's MSISDN, the appropriate method of authentication is selected. If the user has an active Mobile ID account, then the method could be SIM- or App-based digital signature. If the user has issues with his/her Mobile ID account, or has no such account, then the chosen method could be OTP over SMS. For the current use case, the user has an active Mobile ID account, so the flow continues with an attempt for a SIM/App-based digital signature authentication (see chapter 1.3 for more information).
- The AP ID, AP password, signature profile and the DTBS are extracted and prepared from the client's configuration.

Step 8 - The RIG service connects to Mobile ID and sends the MSS Signature Request. The operation is async, so it receives an instant response along with an AP_TransID to use in subsequent polls.

Step 9 - The Mobile ID service sends a signature request to the user's mobile device (via SIM or mobile app).

Step 10 - The user confirms the transaction and authenticates via PIN or fingerprint. A digital signature is created on the mobile device and returned to Mobile ID.

Step 11 - In one of its MSS Status Query polls, the RIG service receives the response with a finished signature status and the digital signature content.

Step 12 - After validating the signature, the RIG service decides to accept the user's authentication request. At this point there are no further checks performed (e.g., proper user authorization) as the RIG service is not in the position of authorizing the user. Therefore, it creates a RADIUS *Access-Accept* packet with an access reply message (created from client's configuration) and sends the packet back to the RADIUS client that called it.

```
Code: Access-Accept
Identifier: 21
Authenticator: cc76edab453b554cd7e7a
Attributes:
- Reply-Message: Authentication successful!
```

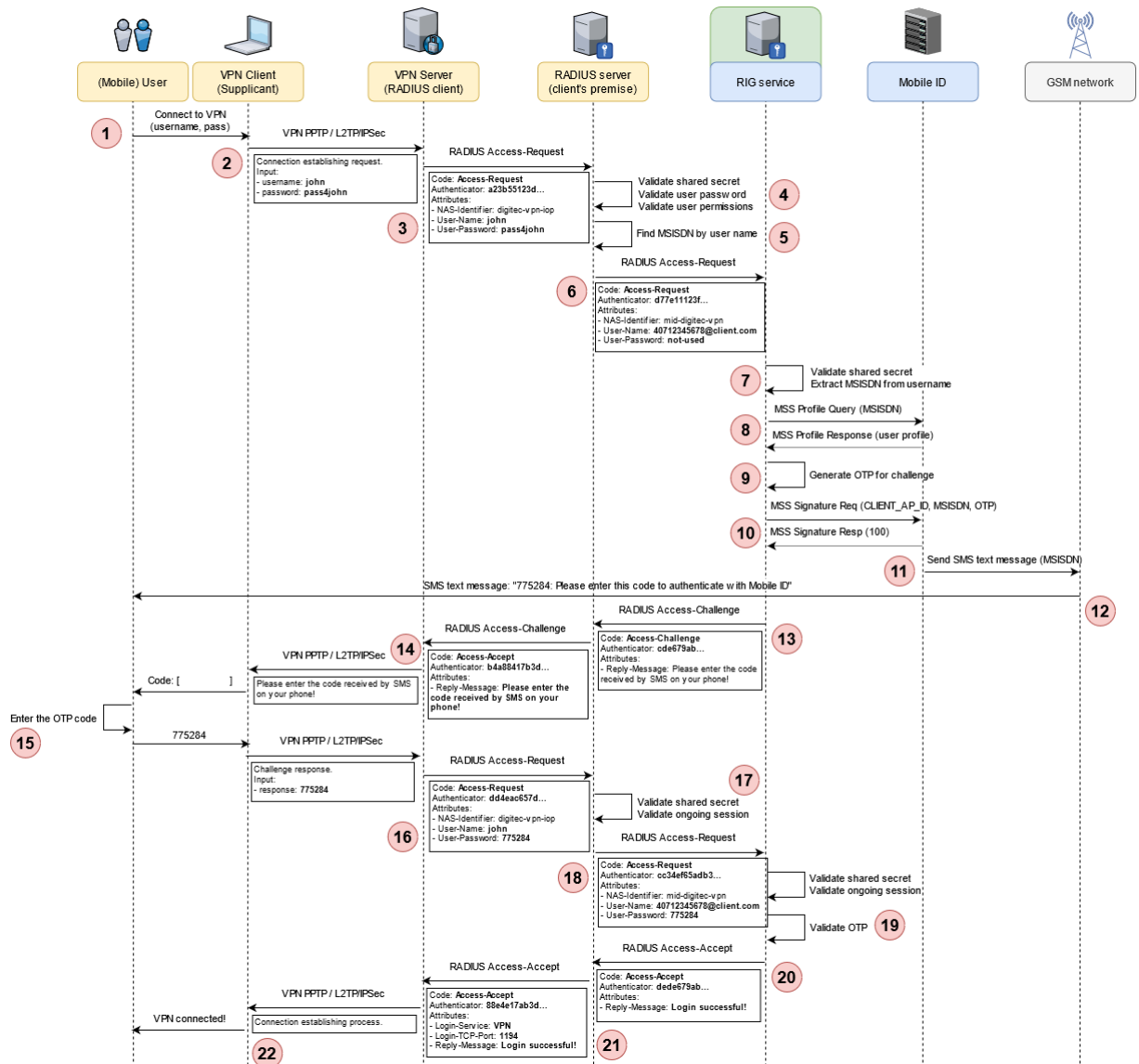
Step 13 - The RADIUS client (i.e., the client's RADIUS server) combines RIG service's access accept packet with its internal evaluation of user's access request and decides to access the user's request, too. A new RADIUS *Access-Accept* packet is created and returned to the VPN server.

```
Code: Access-Accept
Identifier: 15
Authenticator: ec89a776d8cdea8675ac7
Attributes:
- Login-Service: VPN
- Reply-Message: Authentication successful!
```

Step 14 - The VPN server receives the access accept packet and proceeds to provide the user the requested service (i.e., establishes the VPN connection).

2.5.2 VPN Authentication, MFA, SMS OTP

For mobile users that do not have a Mobile ID account or have issues with their account, the RIG service can fallback to a less secure authentication method that uses a one-time password (OTP) generated by the RIG service, sent to the mobile user via an SMS text message and then requested back in a RADIUS Access-Challenge round trip. This method can function both as a fallback method and as a main method of authentication, depending on the RADIUS client configuration at RIG service level.



Step 1 - The user opens the VPN client (the Supplicant) on his/her laptop and clicks on the desired VPN connection. Since he/she does not want to store credentials locally, the VPN client asks him/her to enter the username and the password of the account. The user enters the requested credentials.

Step 2 - The VPN client initiates a specific protocol connection (e.g., PPTP, L2TP/IPSec, etc) to the VPN server, transmitting the username and the password on the line. The VPN server is configured to use RADIUS as authentication protocol.

Step 3 - The VPN server creates a RADIUS *Access-Request* packet using its RADIUS shared secret, the username and password received from the VPN client and the type of service that the user requested: VPN. The RADIUS packet is sent to the client's backend RADIUS server.

```
Code: Access-Request
Authenticator: a23b55123d...
Attributes:
- NAS-Identifier: digitec-vpn-iop
- User-Name: john
- User-Password: pass4john
```

Step 4 - The RADIUS server validates all the aspects concerning the received request: the shared secret and the user password and permissions. It also checks if there are other extra authentication steps required for this user. It finds that this user has multi-factor authentication (MFA) enabled via Mobile ID.

Step 5 - To use Mobile ID as MFA, the flow needs to go through the RIG service. For this to happen, the RADIUS server needs the MSISDN of the currently authenticating user. It makes a local query to its data store and finds the MSISDN of the user based on the given username.

An alternative at this step is for the mobile users to directly enter the username as a construct of "<msisdn>@<domain>" (e.g., "40712345678@client.com") and directly give that value to the VPN client in Step 1 above. Moreover, as per RADIUS flow, this username is also paired to a matching password, therefore having the user enter a wrong (by typo or intentionally) MSISDN is unlikely

Step 6 - The RADIUS server creates a second RADIUS *Access-Request* packet, like the one in **Step 3**. In this case, the RADIUS server acts as a client for the next RADIUS server in the chain: the RIG service operated by Swisscom. The packet created at this step contains the secret shared between this RADIUS server and the RIG service. The username is formatted as "<msisdn>@<domain>" (e.g., "40712345678@client.com"). The user password is not used at this step, so it is filled in with some filler text ("not-used").

```
Code: Access-Request
Authenticator: d77e11123f...
Attributes:
- NAS-Identifier: mid-digitec-vpn
- User-Name: 40712345678@client.com
- User-Password: not-used
```

Step 7 - The RIG service receives the request and proceeds to validate the NAS identifier and the shared secret of the client. Based on the client's configuration, the RIG service decides how to proceed and what parameters to use for the next step:

- Based on the authentication method algorithm configured for this client, an array of possible methods is selected. These can be any of SIM-based digital signature, app-based digital signature, OTP over SMS, or any other method that will be implemented in future versions.
- From the **User-Name** received from the RADIUS client, the user's MSISDN is extracted

Step 8 - Based on an MSS Profile Query for the user's MSISDN, the RIG service selects the OTP over SMS method (see chapter 1.3 for more information).

Step 9 - An OTP code is generated, to function as the challenge request to the user. The content of the challenge message is loaded from the client's configuration.

Step 10 - The RIG service connects to Mobile ID and sends an MSS Signature Request with the signature profile set to "<http://mid.swisscom.ch/MID/v1/OtpProfileText>" (SMS text message via Mobile ID).

Step 11 - On its side, the Mobile ID service connects to the GSM gateway and transmits the SMS text message to the mobile user's device.

Step 12 - The user receives the SMS text message on his/her phone.

Step 13 - After sending the SMS text message, the RIG service assembles a RADIUS *Access-Challenge* response, with the reply message set to inform the user on how to proceed with the challenge (e.g., "Please enter the code received by SMS on your phone!")

```
Code: Access-Challenge
Authenticator: cde679ab...
Attributes:
- Reply-Message: Please enter the code received by SMS on your phone!
```

Step 14 - The RADIUS server passes the *Access-Challenge* response to the VPN server which, in turn, sends the challenge content to the VPN client that runs on the user's machine. The latter one prompts the user to enter the challenge response. In the prompt window, the application displays the challenge text to inform the user how to proceed.

Step 15 - The user checks his/her phone and enter the OTP code received by SMS into the application prompt.

Step 16 - The entered OTP code is sent back point-to-point, from the VPN application to the VPN server and then to the RADIUS server, via a new RADIUS *Access-Request*, which has this time the **User-Password** set to contain the encrypted value of the OTP code.

Step 17 - On the RADIUS server side, the request from the client is again validated using the shared secret and the ongoing authentication session is identified.

Step 18 - The challenge response is packaged again in a new RADIUS *Access-Request* and sent to the RIG service.

Step 19 - On the RIG service's side, the incoming request is validated, and the ongoing authentication session is identified. The decrypted OTP code is checked against the one stored in the authentication session.

Step 20 - Since the received OTP matches the one stored in the authentication session, the RIG service decides to accept the user's authentication request. At this point there are no further checks performed (e.g., proper user authorization) as the RIG service is not in the position of authorizing the user. Therefore, it creates a RADIUS *Access-Accept* packet with an access reply message (created from client's configuration) and sends the packet back to the RADIUS client that called it.

```
Code: Access-Accept
Authenticator: dede679ab...
Attributes:
- Reply-Message: Login successful!
```

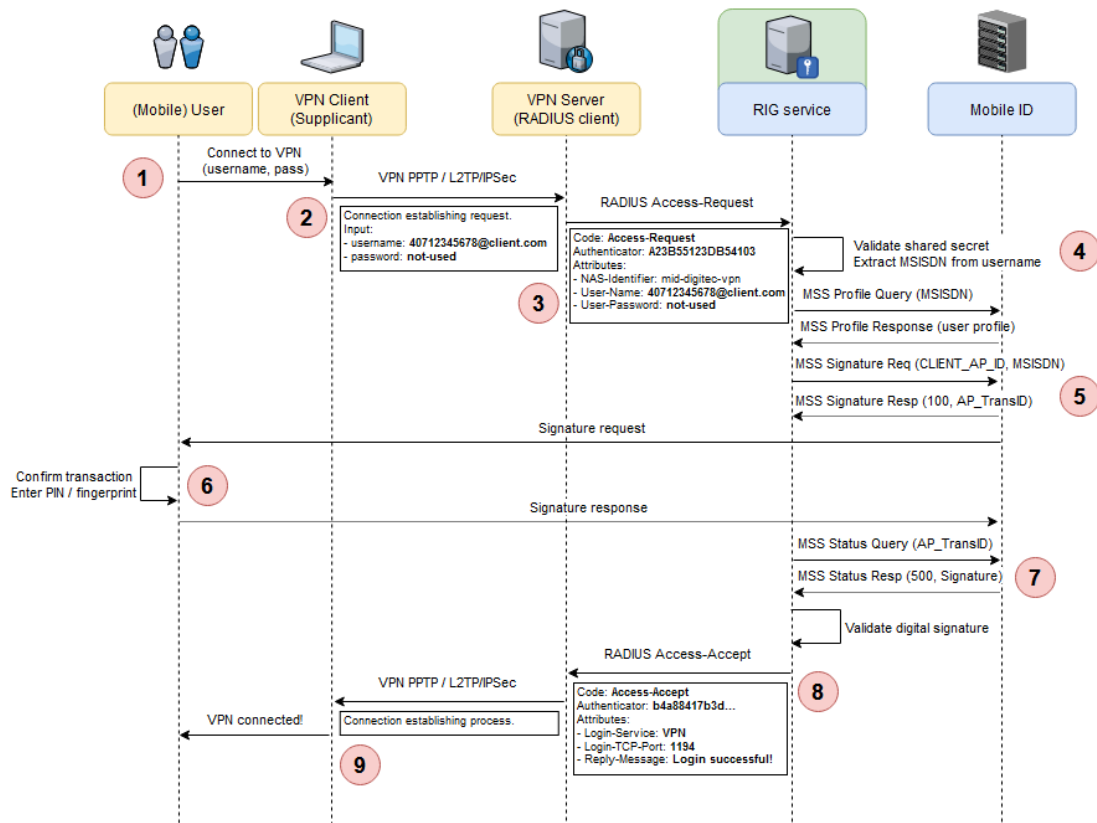
Step 21 - The RADIUS client (i.e., the client's RADIUS server) combines RIG service's access accept packet with its internal evaluation of user's access request and decides to access the user's request, too. A new RADIUS *Access-Accept* packet is created and returned to the VPN server.

```
Code: Access-Accept
Authenticator: 88e4e17ab3d...
Attributes:
- Login-Service: VPN
- Login-TCP-Port: 1194
- Reply-Message: Login successful!
```

Step 22 - The VPN server receives the access accept packet and proceeds to provide the user the requested service (i.e., establishes the VPN connection).

2.5.3 VPN Authentication, SFA, Mobile Signature

For this scenario, the client's architecture relies on a single RADIUS service to perform user authentication: the RIG service. This makes Mobile ID to function as a single factor authentication, in the context of authentication steps that users must go through to obtain access to the desired services.



Step 1 - The user opens the VPN client (the Supplicant) on his/her laptop and clicks on the desired VPN connection. Since he/she does not want to store credentials locally, the VPN client asks him/her to enter the username and the password of the account. The user enters the username in the form of "<MSISDN>@<domain>" and leaves the password field empty (if the VPN client allows it; otherwise, a random password can be entered, as it is not used for this scenario).

Step 2 - The VPN client initiates a specific protocol connection (e.g., PPTP, L2TP/IPSec, etc) to the VPN server, transmitting the username and the password on the line. The VPN server is configured to use RADIUS as authentication protocol.

Step 3 - The VPN server creates a RADIUS *Access-Request* packet using its RADIUS shared secret, the username and password received from the VPN client and the type of service that the user requested: VPN. The RADIUS packet is sent to the RIG service for processing.

```

Code: Access-Request
Authenticator: A23B55123DB54103
Attributes:
- NAS-Identifier: mid-digitec-vpn
- User-Name: 40712345678@client.com
- User-Password: not-used
  
```

Step 4 - The RIG service receives the request and proceeds to validate the NAS identifier and the shared secret of the client. Based on the client's configuration, the RIG service decides how to proceed and what parameters to use for the next step:

- Based on the authentication method algorithm configured for this client, an array of possible methods is selected. These can be any of SIM-based digital signature, app-based digital signature, OTP over SMS, or any other method that will be implemented in future versions.
- From the **User-Name** received from the RADIUS client, the user's MSISDN is extracted
- Based on an MSS Profile Query for the user's MSISDN, the appropriate method of authentication is selected. If the user has an active Mobile ID account, then the method could be SIM- or App-based digital signature. If the user has issues with his/her Mobile ID account, or has no such account, then the chosen method could be OTP over SMS. For the current use case, the user has an active Mobile ID account, so the flow continues with an attempt for a SIM/App-based digital signature authentication (see chapter 1.3 for more information).
- The AP ID, AP password, signature profile and the DTBS are extracted and prepared from the client's configuration.

Step 5 - The RIG service connects to Mobile ID and sends the MSS Signature Request. The operation is async, so it receives an instant response along with an AP_TransID to use in subsequent polls. On its side, the Mobile ID service sends a signature request to the user's mobile device (via SIM or mobile app).

Step 6 - The user confirms the transaction and authenticates via PIN or fingerprint. A digital signature is created on the mobile device and returned to Mobile ID.

Step 7 - In one of its MSS Status Query polls, the RIG service receives the response with a finished signature status and the digital signature content.

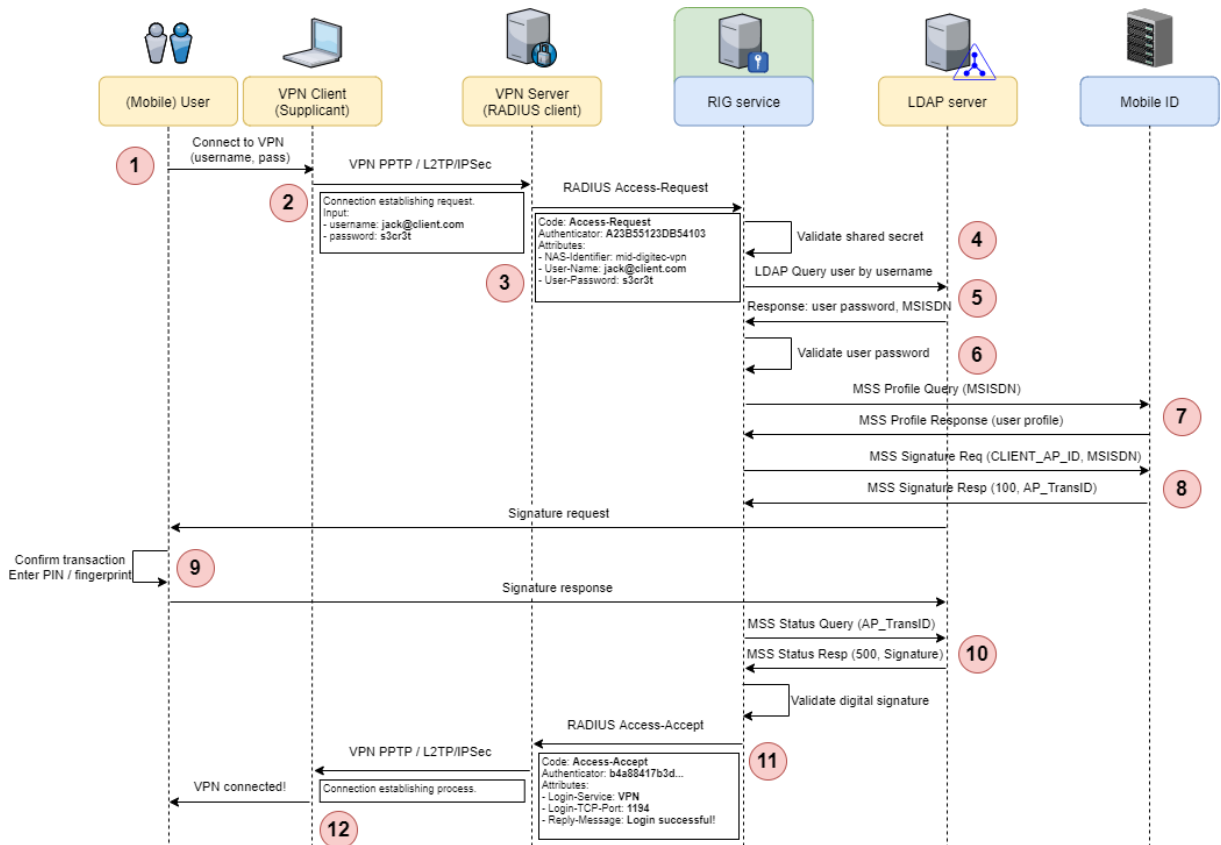
Step 8 - After validating the signature, the RIG service decides to accept the user's authentication request. At this point there are no further checks performed (e.g., proper user authorization) as the RIG service is not in the position of authorizing the user. Therefore, it creates a RADIUS *Access-Accept* packet with an access reply message (created from client's configuration) and sends the packet back to the RADIUS client (i.e., VPN server) that called it.

```
Code: Access-Accept
Identifier: 15
Authenticator: ec89a776d8cdea8675ac7
Attributes:
- Login-Service: VPN
- Reply-Message: Authentication successful!
```

Step 9 - The VPN server receives the access accept packet and proceeds to provide the user the requested service (i.e., establishes the VPN connection).

2.5.4 VPN Authentication, MFA with local LDAP, Mobile Signature

This scenario presents a setup discussed in chapter 2.7, where the RIG service is configured as the only RADIUS server in a client's network. It uses a local LDAP directory instance for validating the user's credentials and the Mobile ID service for second factor authentication via a mobile signature. Moreover, for this scenario the MSISDN is not sent as part of the RADIUS User-Name attribute but it is resolved by the RIG service via the LDAP directory.



Step 1 - The user opens the VPN client (the Supplicant) on his/her laptop and clicks on the desired VPN connection. Since he/she does not want to store credentials locally, the VPN client asks him/her to enter the username and the password of the account. The user enters the username (e.g. "jack@client.com") and the password.

Step 2 - The VPN client initiates a specific protocol connection (e.g., PPTP, L2TP/IPSec, etc) to the VPN server, transmitting the username and the password on the line. The VPN server is configured to use RADIUS as authentication protocol.

Step 3 - The VPN server creates a RADIUS *Access-Request* packet using its RADIUS shared secret, the username and password received from the VPN client and the type of service that the user requested: VPN. The RADIUS packet is sent to the RIG service for processing.

```

Code: Access-Request
Authenticator: A23B55123DB54103
Attributes:
- NAS-Identifier: mid-digitec-vpn
- User-Name: jack@client.com
- User-Password: s3cr3t
    
```

Step 4 - The RIG service receives the request and proceeds to validate the NAS identifier and the shared secret of the client.

Step 5 - Based on the client's configuration, the RIG service performs a query on the configured LDAP directory, retrieving the user password and MSISDN based on the received user-name.

Step 6 - The retrieved LDAP data is used in this step to perform the first factor authentication: validate the user supplied password.

Step 7 - Based on an MSS Profile Query for the user's MSISDN, the appropriate method of authentication is selected. If the user has an active Mobile ID account, then the method could be SIM- or App-based digital signature. If the user has issues with his/her Mobile ID account, or has no such account, then the chosen method could be OTP over SMS. For the current use case, the user has an active Mobile ID account, so the flow continues with an attempt for a SIM/App-based digital signature authentication (see chapter 1.3 for more information). The AP ID, AP password, signature profile and the DTBS are extracted and prepared from the client's configuration.

Step 8 - The RIG service connects to Mobile ID and sends the MSS Signature Request. The operation is async, so it receives an instant response along with an AP_TransID to use in subsequent polls. On its side, the Mobile ID service sends a signature request to the user's mobile device (via SIM or mobile app).

Step 9 - The user confirms the transaction and authenticates via PIN or fingerprint. A digital signature is created on the mobile device and returned to Mobile ID.

Step 10 - In one of its MSS Status Query polls, the RIG service receives the response with a finished signature status and the digital signature content.

Step 11 - After validating the signature, the RIG service decides to accept the user's authentication request. At this point there are no further checks performed (e.g., proper user authorization) as the RIG service is not in the position of authorizing the user. Therefore, it creates a RADIUS *Access-Accept* packet with an access reply message (created from client's configuration) and sends the packet back to the RADIUS client (i.e., VPN server) that called it.

```
Code: Access-Accept
Identifier: 15
Authenticator: ec89a776d8cdea8675ac7
Attributes:
- Login-Service: VPN
- Reply-Message: Authentication successful!
```

Step 12 - The VPN server receives the *Access-Accept* packet and proceeds to provide the user with the requested service (i.e., establishes the VPN connection).

2.6 SMS notifications

The most secure authentication method for users is to use Mobile ID with SIM- or app- based digital signatures. This, however, requires an active Mobile ID account, which is something that users might not have immediately available during a RADIUS authentication session. For this reason, after an authentication session is finished successfully (say, with a fall back to OTP over SMS, where the OTP is correctly provided by the user), the RIG service will follow up to the respective user with an SMS notification for helping the user kick start the process for fixing the issue that he/she has with the Mobile ID account (or create an account altogether).

The SMS text that will be sent cannot fix that much by itself. Also, the RIG service cannot provide the UI required for fixing any account issues or guide the user into creating a new Mobile ID account, as the RIG service is not really the place where such a UI should be implemented. For these reasons, the SMS text that is sent as a delayed notification will contain a friendly text adapted to the respective account situation and a link to the Mobile ID user portal where the user can create a new account or fix any existing issues, he/she has with it.

For each RADIUS client configured for the RIG service, the following delayed notifications will be available:

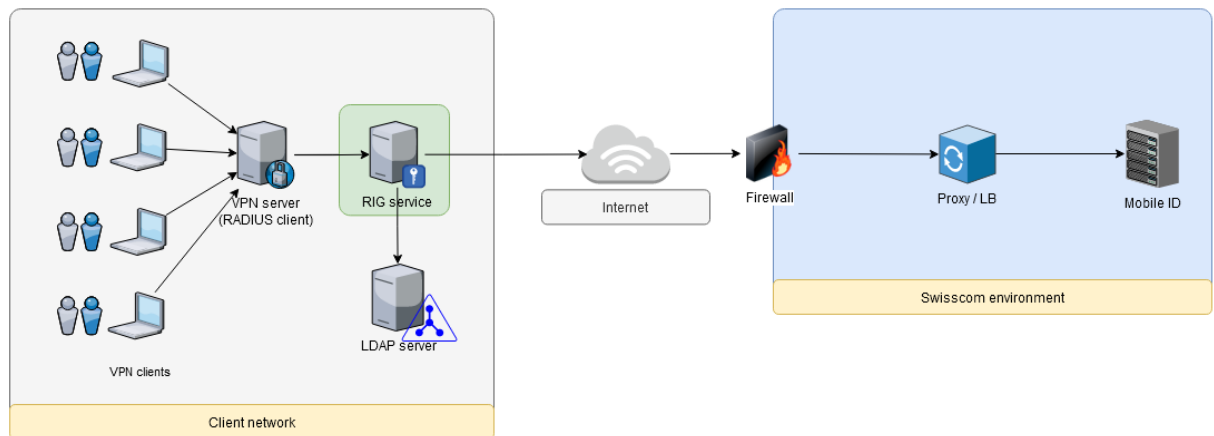
- No Mobile ID account => *"Create a new Mobile ID account by visiting the Mobile ID portal. <https://link>"*
- Account present, not active => *"You can activate your Mobile ID account by visiting the Mobile ID portal. <https://link>"*
- Account present, PIN blocked => *"We noticed your Mobile ID account has its PIN blocked. Visit the Mobile ID portal to unblock it. <https://link>"*

These messages will be configurable on a per client basis, including the template for them (for each of the 4 supported Mobile ID languages), the time between notifications and the total number of SMS messages to send per user, to prevent user spamming.

2.7 LDAP authentication

In the previous chapters we discussed various service setups, one of which could have the RIG service installed on client's premise (either physical premise or client's managed cloud environment). For this setup, if the RIG service needs to act as the main (or only) RADIUS server, it needs to perform both username + password authentication and the second factor authentication via Mobile ID. To accomplish this, an option is to enable the RIG service to validate user's credentials against an LDAP server.

The following diagram presents this setup:



By configuration, the RIG service can connect to a local LDAP server and validate the received RADIUS **User-Name** and **User-Password** against a set of LDAP user attributes. Once this validation is successful, the service can move to the Mobile ID-based authentication part of the flow.

This feature allows the RIG service to be a quick drop-in and replace the existing RADIUS server for clients that are already having this combination (RADIUS + LDAP database).

3 The RADIUS Protocol

3.1 Remote Authentication Dial-In User Service (RADIUS) Protocol

The RADIUS protocol, which stands for "Remote Authentication Dial-In User Service" is a network protocol that controls user network access via authentication, authorization, and accounting (AAA or Triple A). It is commonly used for allowing users to access various network devices and services while providing a centralized user and permission management.

The protocol is usually hidden inside controlled network and service access software and it is not seen directly by end users. It is used only during the session-establishing part of a network traffic. The actual point-to-point communication (the network use that the users are requesting) is not flowing through the RADIUS server and it is not part of the RADIUS protocol.

Ease of use and flexibility are the main characteristics of the RADIUS protocol. It can be easily implemented by network devices, access services and terminal servers while, at the same time, can be extended and enriched with custom validation and authentication schemes. The latter feature fits nicely with the scope of this document.

3.1.1 Protocol Overview

The RADIUS protocol is a binary network protocol, operating on ports 1812 and 1813 and running, with respect to the network stack, in the application layer, on top of TCP and UDP. The most common implementations of RADIUS are using the UDP network protocol, with TCP connections being also supported.

The protocol is covered by several RFCs, the most important ones being:

- [RFC 2865 - Remote Authentication Dial-In User Service \(RADIUS\)](#)
- [RFC 2866 - RADIUS Accounting](#)
- [RFC 2869 - RADIUS Extensions](#)
- [RFC 6613 - RADIUS over TCP](#) (updated by [7930](#))

A larger list of related RFCs can be found in Annex 1.

3.1.2 Key Concepts and Terminology

The following important concepts and terms are used in this document when referring to the RADIUS protocol:

Element	Description
User	The person (or system) that uses a Supplicant to request access to a network service (or other type of service). In the Mobile ID world, this would be a mobile user.
Supplicant	The software that the User is employing to access the network service. The Supplicant collects the credentials from the User and connects to a NAS for requesting access for the User and establishing the service connection.
NAS	Network Access Server. Although it has "server" in its name, the NAS is the network application/equipment that acts as a client to a RADIUS server. Being asked by the Supplicant to authenticate the User and permit network/service access, the NAS connects then to a pre-configured RADIUS server and asks for user authentication and authorization. NAS-es include FTP servers, Web servers, Unix login services, VPN servers, remote desktop servers, etc.
RADIUS server	The central server component that decides which Users can access what network services. It is generally one central point in a RADIUS architecture, being connected to all NAS-es that Users can access.
Authentication Session	An authentication session is a communication between User - Supplicant - NAS and one or more RADIUS server(s) with the scope of permitting access to the User to a network service. The session is always initiated by the User and can result in a permission or in a rejection.

Being a binary protocol over UDP, the RADIUS packets sent back and forth between the client and the server are composed of bytes with a clearly defined yet flexible structure for representing the request or response data. The complete packet structure can be found in [RFC 2865, section 3. Packet Format](#).

For the current document, the important aspect of a RADIUS packet are the fields inside and the values that these fields have:

- **Code:** identifies the type of RADIUS packet. While the values are in the range of 1 byte (0-255), for the current document we will refer to the respective semantic values: *Access-Request*, *Access-Accept*, *Access-Reject*, *Access-Challenge* (there are more values that can be used for the Code field, corresponding to other packet types, but, for the current discussion, these 4 are the most important ones).
- **Identifier:** unique number (1 byte) used for matching requests and responses, for the same client source IP address and client source UDP port, in a short span of time.
- **Authenticator:** (16 bytes) contains the salt-like vector that is used, together with the shared secret and the user's entered password, to create a unique hash that helps the RADIUS server to authenticate the calling client.
- **Attributes:** the attributes of the packet. These depend on the packet type and the service that the user has requested. Logically, they represent the payload of the packet, with the 3 above being the metadata of the packet.

The complete reference for the attributes that a packet can contain is available in the [RFC 2865, section 5](#).

Since the structure of a packet is a stream of bytes, the attributes must be encoded in a Type - Length - Value (TLV) form. For the current discussion and to ease the formatting of packet content, the attributes will be represented as a list of name and values, with just the most relevant attributes being included in each snippet.

Here is a list of attributes that will be used in the next chapters of the document:

- **User-Name:** the name of the user to be authenticated. It must be sent in *Access-Request* packets.
- **User-Password:** the password of the user to be authenticated. It is only used in *Access-Request* packets.
- **Reply-Message:** indicates a text that may be displayed to the user. It can be used in an *Access-Accept* (the success message), *Access-Reject* (the failure message), *Access-Challenge* (a text to be presented to the user for the challenge).
- **State:** 3 or more characters, representing a correlation token that must be sent back, unchanged, when the client needs to send to the RADIUS server a response to an *Access-Challenge* request.
- With these fields and attributes defined, in the next chapter an *Access-Request* packet will be written as:

```
Code: Access-Request
Identifier: 10
Authenticator: A23B55123DB54103
Attributes:
- NAS-Identifier: digitec-vpn
- User-Name: john
- User-Password: md5("secret"+Authenticator)
```

a subsequent *Access-Accept* packet will be written as:

```
Code: Access-Accept
Identifier: 10
Authenticator: b4a88417b3d0170d754c647c30b7216a
Attributes:
- Login-Service: Telnet
- Login-TCP-Port: 8080
- Reply-Message: Authentication successful!
```

and, finally, an *Access-Reject* packet reply will be written as:

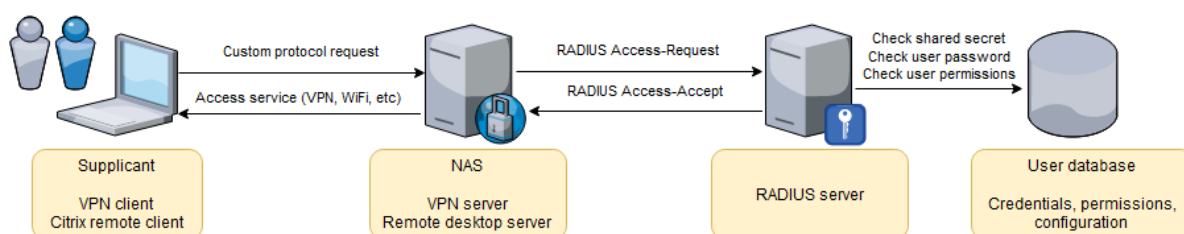
```
Code: Access-Reject
Identifier: 10
Authenticator: b4a88417b3d0170d754c647c30b7216a
Attributes:
- Reply-Message: Invalid credentials!
```

3.1.3 Basic Flow Description

A typical RADIUS authentication session is initiated by the user, requesting a specific service to a device or software that is installed on, or accessible from, the user's machine. Such software, called the Supplicant, collects the credentials of the user (username and password; these might also be stored for future use, so that the user does not need to enter them each time) and sends them via its own protocol to a Network Access Service (NAS) component, capable of providing the requested service. This NAS component is configured to use RADIUS for user authentication, so it creates a RADIUS *Access-Request* packet, containing the following:

- The user credentials (**User-Name**, **User-Password**)
- An encrypted form of the shared secret between it and the destination RADIUS server.
- The NAS client IP and port
- The service type that the user requested
- Any additional attributes that might be needed for proper user authorization

The diagram below presents this exchange:



The configured RADIUS server receives the packet and proceeds to check the validity of the data and authenticate the user. At this point, the behaviour depends on the configuration of the RADIUS server and the process that is required for authenticating and authorizing the user. The RADIUS server might contact other servers to complete the authentication and authorization process and decide on a positive (accept) or negative (reject) response.

In the diagram above, the RADIUS server decided to inform the NAS client that the user's request is OK and that the requested service can be provided. For this, the RADIUS server sends back a RADIUS packet with the code 2 (*Access-Accept*). Should it decide to reject the user's request, the RADIUS server would send back to the NAS client a packet with code 3 (*Access-Reject*).

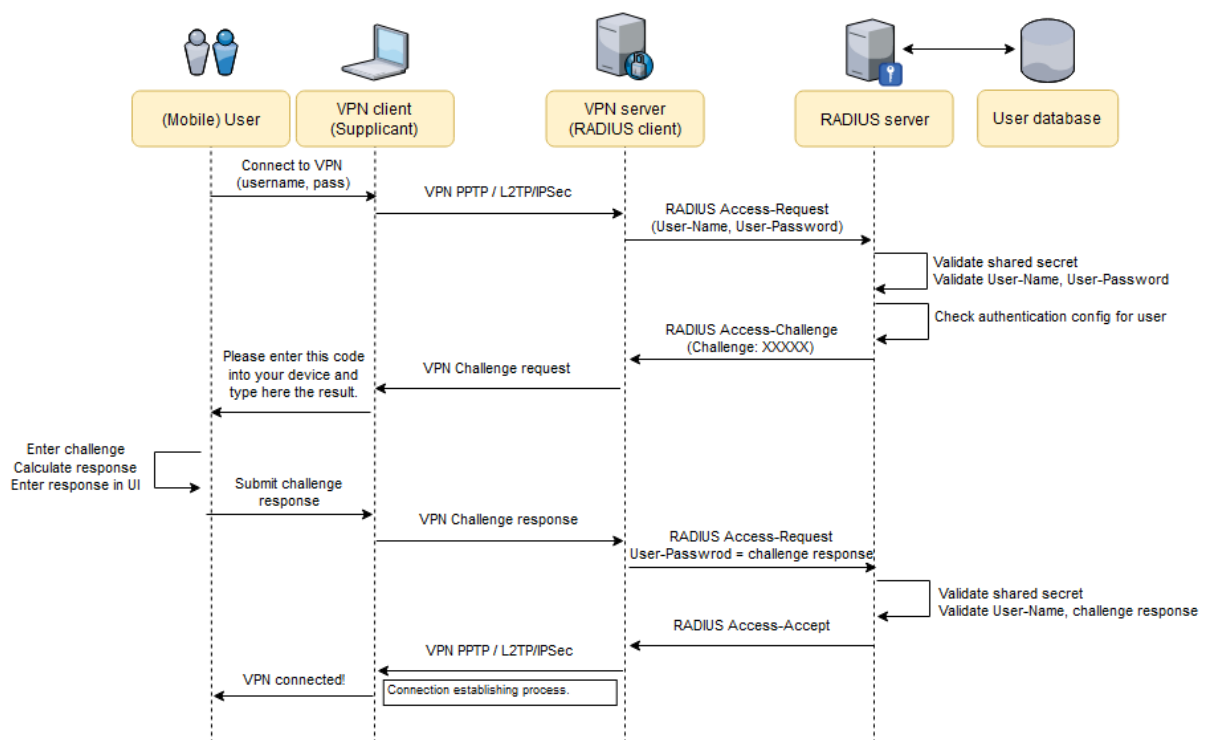
As the final step in our flow, the NAS client receives the response packet from the RADIUS server and, if it has an *Access-Accept* code, it moves on to providing the requested service (e.g., VPN connection establishing).

3.1.4 RADIUS Challenge and Response

Based on the configuration for a user and during an authentication session, the RADIUS server can decide to perform a challenge/response authentication. This flow introduces a few more steps in the standard RADIUS flow:

- After the RADIUS server receives the first RADIUS *Access-Request*, it sends back to the RADIUS client an *Access-Challenge* response (instead of an *Access-Accept* or *Access-Reject*). The response packet contains a challenge code that the user is expected to enter in a security device (smart card or software application) and obtain a response.
- The RADIUS client receives the *Access-Challenge* response and uses its custom protocol with the Supplicant application to transfer this challenge and present it to the user.
- The Supplicant application (e.g., VPN client application) displays the challenge to the user and instructs him/her to enter the challenge code into his/her security device, calculate the response code and enter that code back in the UI.
- The user performs the computation and enters the code back.
- The Supplicant application sends the response code back to the RADIUS client which, in turn, creates a new RADIUS *Access-Request* packet, this time with the **User-Password** field set to the challenge response.
- The RADIUS server identifies the ongoing authentication session, checks the challenge response, and decides whether to accept the request of the user or not. Depending on the decision, the RADIUS server sends back either an *Access-Accept* (request accepted!), an *Access-Reject* (request rejected!) or another *Access-Challenge* (more challenge roundtrips are required).
- Finally, the Supplicant application and the RADIUS client act together based on the response from the RADIUS server: either give the user access to the service, reject the user, or challenge him/her further.

The diagram below depicts the Challenge/Response flow:



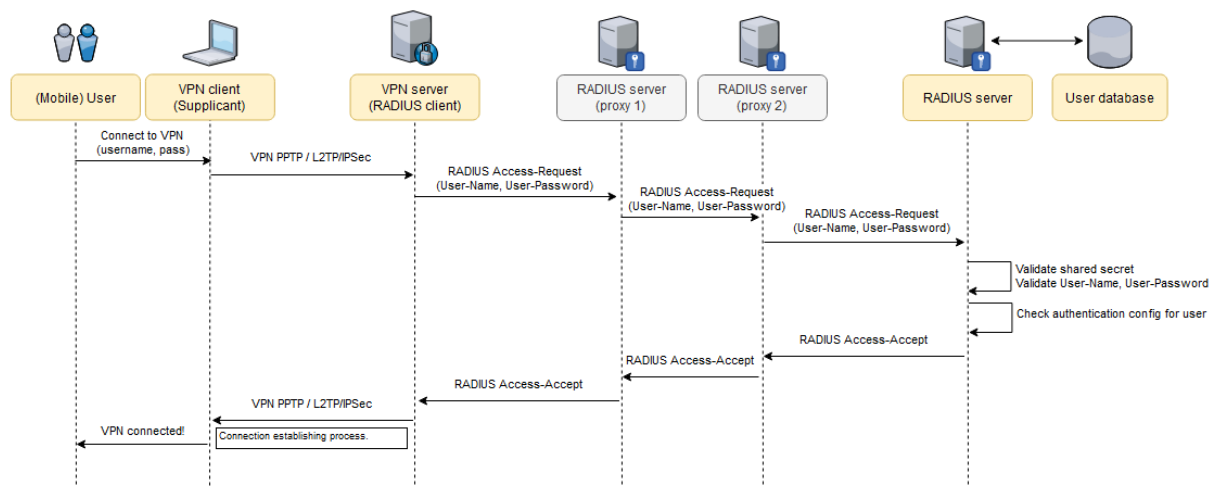
The goal of the RADIUS Challenge/Response flow is to increase the strength of the authentication process by using a two-factor authentication: **User-Name** + **User-Password** (something the user knows) and the challenge code calculation (something the user has => the security device).

3.1.5 RADIUS via Proxies

The RADIUS protocol allows network architectures where certain RADIUS server components are acting as proxies between a RADIUS client and a remote (final) RADIUS server. Whether this is for roaming/federation purposes or for enhancing an authentication session with input from more than one RADIUS server, using RADIUS proxies is an easy and transparent way of assembling a custom authentication flow:

- The authentication session starts as usual, with the user requesting a particular service to a Supplicant application. The request is sent via a specific protocol to the backend service that acts as a RADIUS client in this case.
- The RADIUS client assembles an *Access-Request* packet and sends it to the configured RADIUS server.
- Based on the configuration for this RADIUS client or user, the RADIUS server decides to proxy the request, so it sends the *Access-Request* to the next RADIUS server. For the diagram below, this server is again a proxy, so this step is repeated once more.
- The *Access-Request* packet finally lands on the remote RADIUS server. After the due security checks and authorizations, the remote RADIUS server can issue any of an *Access-Accept*, *Access-Reject* or *Access-Challenge*.
- The response travels back, from service to service, in reverse order, until it reaches the RADIUS client. At this point, the client acts based on the received response type (accepts or rejects the service or takes the user through a challenge).

The following diagram depicts this scenario:



The Proxy scenario is a good asset for assembling a RADIUS-based network. It allows the administrators to change the topology of the network without affecting existing RADIUS clients (e.g., VPN server endpoints), existing applications installed on user's machines or the users' current behaviour.

4 Annexes

4.1 Annex 1 - List of RADIUS-related RFCs

The following list contains a set of RFCs that are covering the various aspects of the RADIUS protocol and that will come handy for implementing the solution outlined in this document:

- [RFC 2865 - Remote Authentication Dial-In User Service \(RADIUS\)](#)
- [RFC 2866 - RADIUS Accounting](#)
- [RFC 2867 - RADIUS Accounting Modifications for Tunnel Protocol Support](#)
- [RFC 2868 - RADIUS Attributes for Tunnel Protocol Support](#)
- [RFC 2869 - RADIUS Extensions](#)
- [RFC 3162 - RADIUS and IPv6](#)
- [RFC 3575 - IANA Considerations for RADIUS](#)
- [RFC 3579 - RADIUS Support for Extensible Authentication Protocol](#)
- [RFC 2580 - IEEE 802.1X RADIUS Usage Guidelines](#)
- [RFC 5080 - Common RADIUS Implementation Issues and Suggested Fixes](#)
- [RFC 6158 - RADIUS Design Guidelines](#)
- [RFC 6572 - RADIUS Support for Proxy Mobile IPv6](#)
- [RFC 6613 - RADIUS over TCP \(updated by 7930\)](#)
- [RFC 6614 - Transport Layer Security \(TLS\) Encryption for RADIUS](#)
- [RFC 6929 - RADIUS Protocol Extensions](#)
- [RFC 7268 - RADIUS Attributes for IEEE 802 Networks](#)
- [RFC 7930 - Larger packets for RADIUS over TCP](#)
- [RFC 8044 - Data Types in RADIUS](#)
- [RFC 2607 - Proxy Chaining and Policy Implementation in Roaming](#)

4.2 Annex 2 - RADIUS Testing tools

For testing a running instance of the RIG service (or any other RADIUS server, really), the following tools can be used.

4.2.1 Radclient - command line tool

The [Radclient](#) is a small RADIUS client program that can be used from the command line to send RADIUS packets and print the received responses. Input data can be given via program arguments or with a local configuration file.

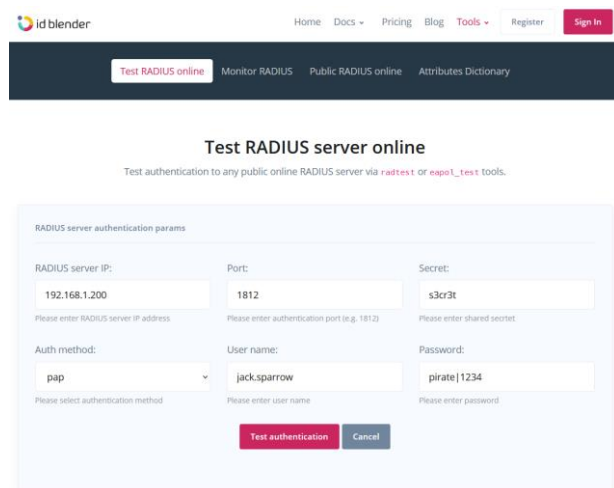
It can be used like this:

```
echo "User-Name = test" | /usr/local/bin/radclient localhost:1812 auth s3cr3t
echo "User-Name=test,User-Password=mypass,Framed-Protocol=PPP" | \
  /usr/local/bin/radclient localhost:1812 auth s3cr3t
echo "Message-Authenticator = 0x00" | /usr/local/bin/radclient localhost:1812 auth s3cr3t
```

4.2.2 RADIUS test online application

The [idBlender](#) company provides a free online web application that functions as a RADIUS client. It uses a backend service to perform the actual RADIUS request (so it is not the browser that is sending the RADIUS requests, but a backend service) and can be used to test publicly available RADIUS servers.

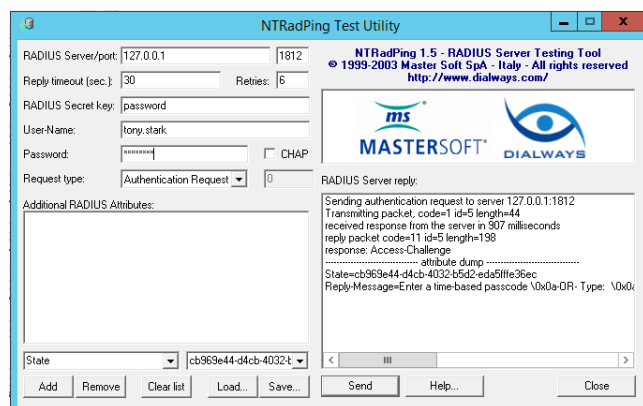
Here is a screenshot from the application:



4.2.3 NTRadPing Test Utility - Windows desktop application

The [NTRadPing](#) application is a Windows desktop application that can be used for testing a RADIUS server. Since it runs from a local machine, it can easily test any internal/private RADIUS service.

Here is a screenshot from the application:



4.2.4 Other tools

There are other tools that could come handy during the development and testing phases of the RIG service. For example, the [Simple Radius Test Tool](#) is an ad-supported Android application that functions as a RADIUS client, and [RadPerf](#) is a load testing tool for RADIUS servers.