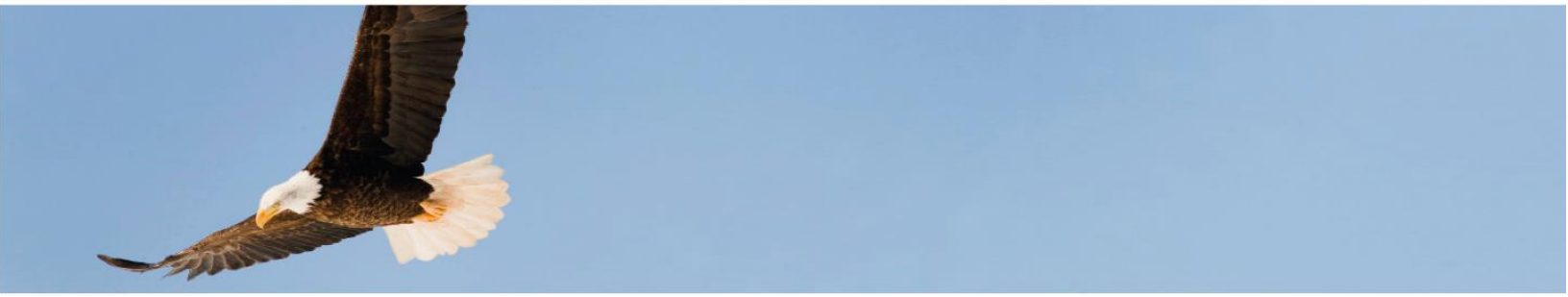
Salt Group



Echidna
Administrator Reference

Version 15.2

October 2015

© 2015 Salt Group Proprietary Limited. All rights reserved.

Information in this document is subject to change without notice. The software described in this document is furnished under a license agreement or nondisclosure agreement. Copies of software supplied by Salt Group must not be released to any party without written authorisation from Salt Group and remain the property of Salt Group for all time. They may not be transferred to any computer without both a service contract for the use of the software on that computer being in existence and written authorisation from Salt Group.

Copies of software released by Salt Group to academic establishments may not be used for any commercial purpose without written authorisation from Salt Group and royalty payments made to the company.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Salt Group.

Whilst Salt Group has made every effort in the preparation of this manual to ensure the accuracy of the information, the information contained in this manual is delivered without warranty, either express or implied. Salt Group will not be held liable for any damages caused, or alleged to be caused, either directly or indirectly by this manual.

Licenses and Trademarks

All other brands and their products are trademarks or registered trademarks of their respective holders and should be noted as such. All other trademarks acknowledged.

Contents

Chapter 1: Before starting

This *Administration Reference* is part of a set of books about Echidna. It describes how to configure and maintain Echidna.

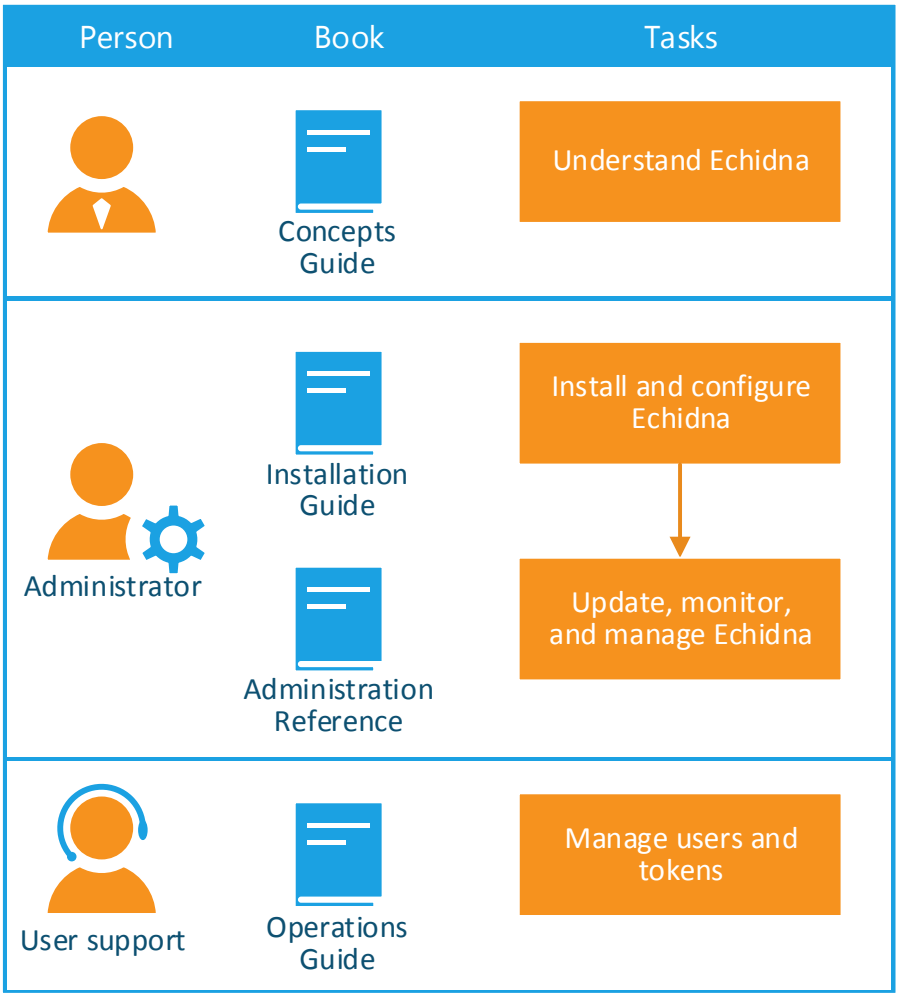


Figure 1: When to use each book in the Echidna documentation set

1.1 About Echidna

Echidna is a multi-factor authentication server that can accept authentication requests from clients through RADIUS and HTTP/S (web services).

Echidna supports multiple authentication methods, including these:

- **Hardware security tokens** that comply with the OATH HOTP, OATH TOTP or OATH OCRA standards
- **Salt Mobile tokens** are standalone OTP mobile apps. These are supported on many mobile devices and platforms, including Android, iOS, Blackberry, and Symbian.
- **One-time passcodes (OTPs)** delivered to a user by SMS or email.
- Echidna is pluggable, to support additional authentication methods.

Echidna can broker authentication requests to a legacy authentication server, such as RSA Authentication Manager. It does this by proxying the authentication requests to third-party servers to support legacy or proprietary tokens, such as RSA SecurID and Vasco tokens.

Echidna authenticates users from existing user stores. Alternatively, Echidna can store user records in its own database.

Echidna is supplied as a virtual appliance. The *Installation Guide* describes how to set this up.

1.2 Components of Echidna

Echidna consists of a central server, and three web applications. This diagram shows the three applications, and the ports used to access the applications and the server:

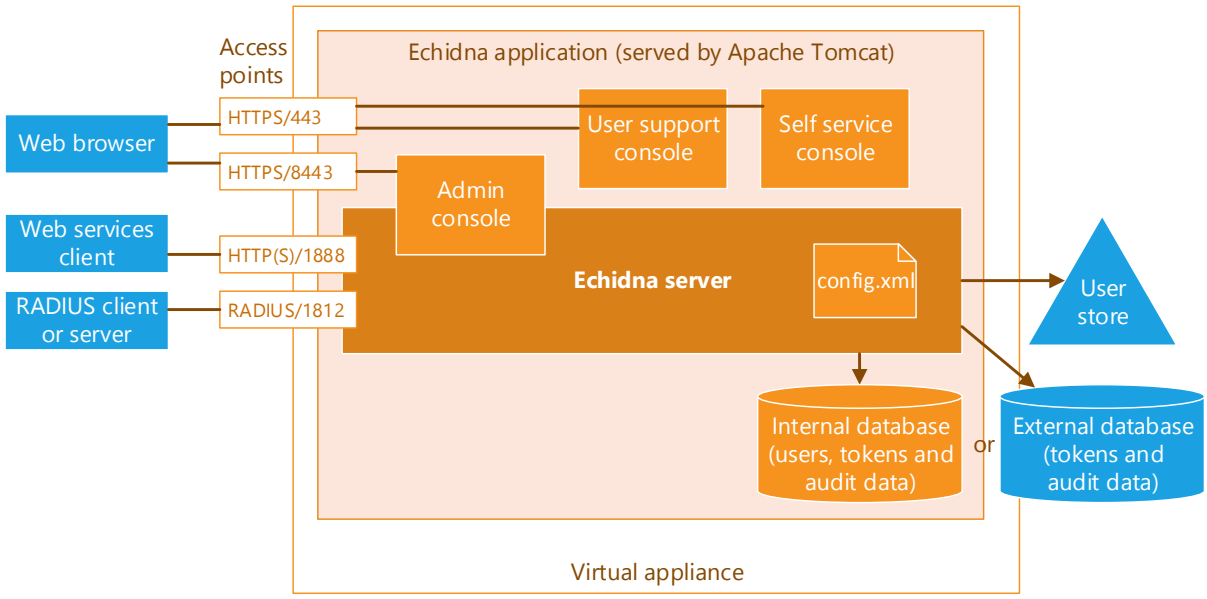


Figure 2: Components of Echidna

1.2.1 Virtual appliance

The Echidna virtual appliance is a virtual machine that already includes Echidna. For instructions about setting up the virtual appliance, see the *Echidna Installation Guide*.

1.2.2 Echidna application

The Echidna application is hosted by Apache Tomcat 7. This application includes the Echidna server, three web applications, and an internal Derby database.

It is also possible to install Echidna on any application server that complies with Servlet API 2.5 (JSP 2.1). However, this book does not describe how to do this.

1.2.3 Echidna server

The core Echidna service engine defines the user stores, roles, and authentication methods.

1.2.4 Access points

Echidna publishes services on access points that are created by Echidna itself (not by the hosting application server).

Echidna can accept RADIUS and web services requests.

1.2.5 User store and token/audit store

The user store and the token and audit store can be either external stores (as in this diagram) or they can be a database inside Echidna.

1.2.6 Administration console, User Support console, and Self Service console

These web applications are supplied with Echidna:

Name of web application	Described in	Used by	Comments
Administration console	This book	Administrators	Let administrators do these tasks: • Configure Echidna, including connecting to user stores, giving access to the other two web applications, and setting up authentication methods. • Save the entire configuration as XML, and import previously saved configuration. • Examine cached event records.
User Support console	*Operations Guide*	Support staff	Gives access to user and token support tasks, using roles created in the Administration console.
Self service console	*Operations Guide*	End users	Let end users register their own supported tokens and manage their own login preferences.

The console applications use Echidna to authenticate and authorise the users who are permitted to access them. For authorisation, Echidna defines the available roles and the rules that make each user a member or not, then the console applications themselves define the pages that are available for each role. For information about defining which users can access the web applications, see Access to the User Support and Self Service consoles on page 34.

Although the Self Service console is enabled by default, it is useful only if the end users have something to manage, as in these cases:

- Echidna is configured to use the Salt mCodeXpress authentication method. End users can manage Salt mCodeXpress mobile soft tokens in the Self Service console.
- Echidna is configured to use the Yubikey token authentication method. End users can manage Yubikey tokens in the Self Service console.
- Echidna is configured to let users select their preferred authentication method, as described in Add an authentication process on page 49. End users can make this selection in the Self Service console.

1.3 Echidna configuration

Echidna stores its configuration in XML format. Administrators can export this configuration and then import all or some of it.

The Echidna configuration consists of nested and cross-referenced XML elements, which means that it is very flexible. This diagram shows the elements that can be configured:

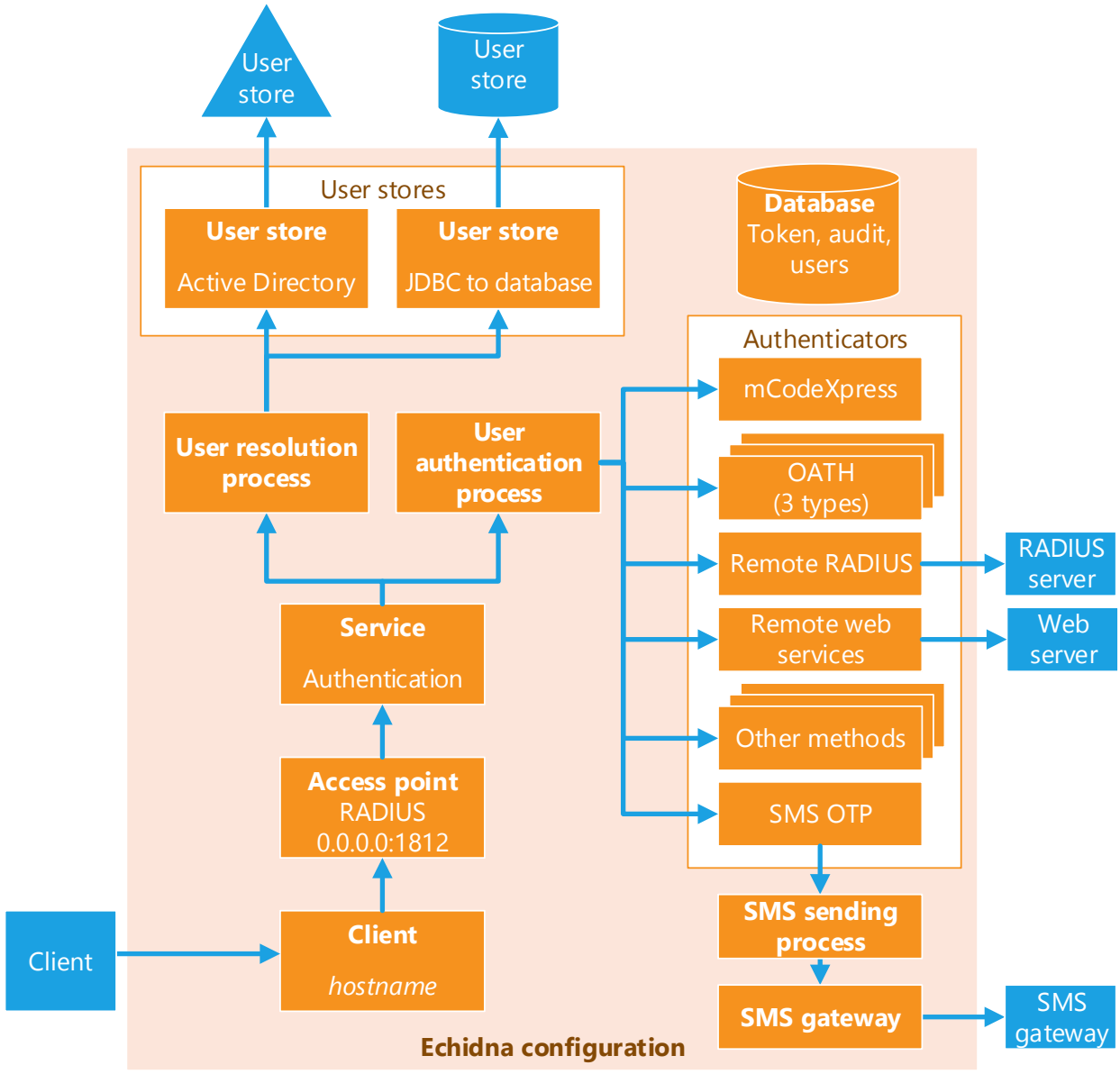


Figure 3: Structure of the Echidna configuration

Two Echidna servers can share the same XML file if it includes both licenses. The keystores must be copied separately.

1.3.1 Order of configuration

Each configuration item in Figure 3 relies on other configuration items. This Administration Reference describes the recommended order for adding to the Echidna configuration.

If an administrator configures items in a different order, they must ensure that their changes work correctly. For example, the list below shows that administrators should link to user stores before adding another user resolution process. When this happens in the recommended order, the user store is automatically included in the new process. However, if the user resolution process is defined and then another user store is added, the administrator must edit the process to include the new user store.

1. Configure the Echidna database (see page 19).

   Echidna requires a database to store token records, audit information, and event information. Echidna can use its own internal Derby database, or it can be configured to use an external database.

   In addition, Echidna can use its internal database as a user store. This is useful if an organisation does not already have a user store.

2. Configure the user data (see page 23).

   If there are no user stores, use the internal database as a user store.

3. Configure a user resolution process. These are the methods that Echidna uses to decide which user information to authenticate against, and how to find the appropriate user record from multiple configured user stores. See page 33.

4. Configure authentication:
   a. Add an authenticator (see page 48). Each authenticator defines an authentication method that Echidna can service.
   b. Add an authentication process (see page 49).

5. Configure services:
   a. Configure services that Echidna offers (see page 105).
   b. Configure access points on which the services is available (see page 107).

6. Configure clients that can request authentication services from Echidna (see page 109).

1.4 Passwords and passcodes

Echidna uses many codes:

- **Password**: The user's code, contained in the user store.
- **Passcode**: A code generated by a token. This is sometimes called a token code.
- **OTP (one-time passcode)**: A single-use passcode that is generated by a token or sent to a user via SMS.
- **Authentication code**: Used for signing, rather than logging in.

Chapter 2: Configure the appliance system

This chapter describes how to update the appliance.

2.1 Open the Echidna appliance console

The Echidna appliance has a console page that lets administrators change some basic aspects of the appliance. This page also lists the URLs for accessing the various web appliances for interacting with Echidna.

The *Installation Guide* describes how to access this page for the first time, and update the default root password to a new secure password. If necessary, ask the person who installed Echidna for this password.

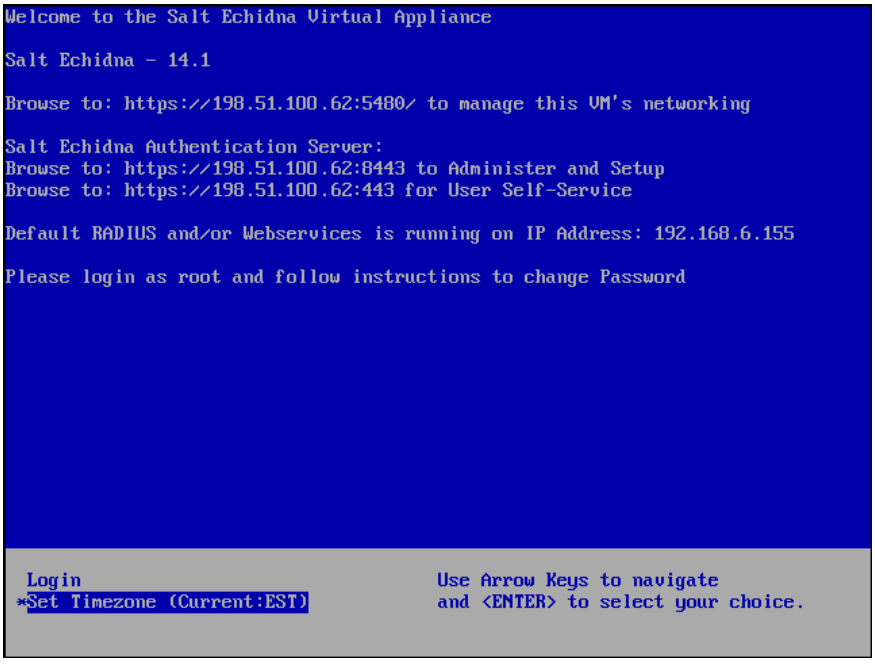


Figure 4: The Echidna appliance console page

2.2 Connecting with SSH

The Echidna appliance can also be accessed by SSH using the **echidna** user. The *Installation Guide* describes how to set a password for this user.

Connecting with SSH gives you access to the same commands available via the console, however you must prefix each command with **sudo** in order to run with elevated privileges.

E.g. `sudo service tomcat start`

2.3 Networking

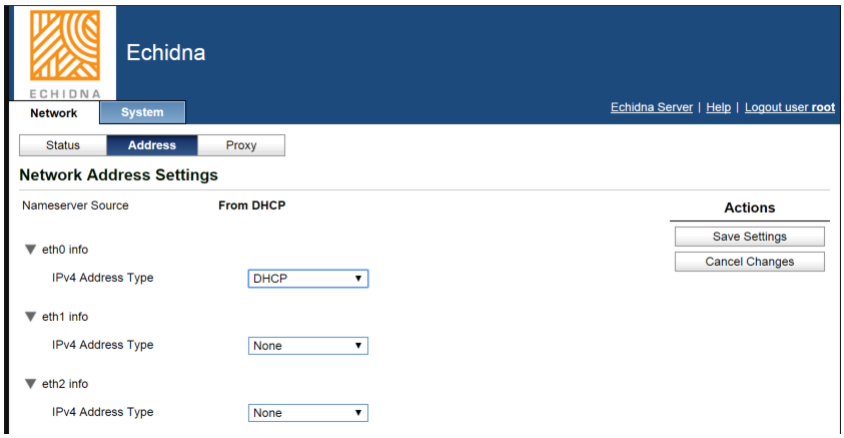
The Echidna appliance comes with a web application that lets administrators manage the appliance itself.

To find this web application, use the first URL listed in the appliance console page. In the example in Figure 4, this URL is **https://198.51.100.62:5480**.

2.3.1 IP addresses

When Echidna is installed, it uses a dynamic IP address. Salt Group recommends that administrators configure Echidna to use a static IP address.

1. In a web browser, open the Echidna appliance management page.
2. Click **Address**. The page shows the following options:



3. Change each Echidna interface to be static.
4. Click **Save Settings**.

2.3.2 DNS Client Configuration

Echidna should be updated to use the network's DNS.

1. Open the Echidna appliance console (see page 13), and log in as **root**.

2. Edit the file /etc/resolv.conf and enter details of the DNS, domain and search names.

   For example:

   ```
   search example.com management.example.com external.example.com
   nameserver 10.0.0.10
   nameserver 8.8.8.8
   ```

2.3.3 Host name

When Echidna is first installed, its hostname is set to **localhost@localdom**. To change this, follow these steps:

1. Open the Echidna appliance console (see page 13), and log in as **root**.
2. Run the following command and replace **localhost@localdom** with the required hostname:

   ```
   # editor /etc/hostname
   ```

3. Run the following command to update the **localhost@localdom** references to the fully qualified hostname:

   ```
   # editor /etc/hosts
   ```

4. Restart Echidna (see page 16).

2.3.4 Network interfaces

The network configuration for Echidna permits one, two, or three network interfaces.

By default, the Echidna appliance has only one network interface, on which all services are available.

This table lists the services that use each interface if Echidna is configured with **three network interfaces**:

Interface	Scoped Identifier Name	Gives access to services
eth0	au.com.saltgroup.radius.adminip	Default network for all services if no other is present. If other networks are present, eth0 is for administration services: • Administration console • User Support console
eth1	n/a	Client services: • Web services requests • RADIUS requests
eth2	au.com.saltgroup.radius.userip	User services: • Self-Service console

2.4 Restart Echidna

Echidna runs inside Apache Tomcat. To restart the Echidna service, reboot the VM.

1. Open the Echidna appliance console (see page 13), and log in as **root**.
2. Stop and start Tomcat with the following commands:

```
service tomcat7 stop
service tomcat7 start
```

2.5 Time zones

The Echidna appliance uses the UTC timezone, until it is changed by an administrator.

To change the time zone:

1. Open the Echidna appliance console (see page 13), and log in as **root**.
2. Type **exit** to return to the main page.
3. Select the **Set Timezone** option.
4. Use the options to select the server's location.
5. Follow the prompts to set the time zone.

2.6 Time synchronisation

Echidna works with the following time-based tokens that rely on Echidna having a good estimate of the token (handset) time when the OTP was generated:

- Salt mCodeXpress
- OATH TOTP
- OCRA with time input

These tokens can continue to work correctly when the server clock drifts slowly, but they can stop working if the clock suddenly jumps too far.

This can happen if the Echidna appliance is migrated to another hypervisor with a different hardware clock.

In addition, some regions must have time synchronization windows set to low values, which can make this a real operational problem.

The *Operations Guide* describes how the operator can use the User Console to resynchronise tokens.

2.6.1 Set the time synchronisation server

Network Time Protocol (NTP) is a protocol to allow for clock synchronization. Machines can be configured to sync their times with an NTP server to ensure their clock is accurate. The machine needs network connectivity in order to request the time from the server.

Add the names of the NTP servers to /etc/ntp.conf. The most important lines start with the **server** directive. By default, the Ubuntu NTP server is configured:

```
server ntp.ubuntu.com
```

To add or replace time servers, edit the file and insert more lines, as in these examples:

```
server 165.187.223.5
server 10.1.254.60
server ntp.ubuntu.com
```

The appliance queries the servers in order from top to bottom.

2.6.2 Get the status of the NTP service

To get the status of the NTP service, run one of the following commands:

```
/etc/init.d/ntp status
service ntp status
```

2.6.3 Check the current drift time

Look at the drift file, which is where NTP writes the current value. It is configured in /etc/ntp.conf and defaults to /var/lib/ntp/ntp.drift:

```
# grep ^driftfile /etc/ntp.conf
driftfile /var/lib/ntp/ntp.drift
# cat /var/lib/ntp/ntp.drift
0.000
```

2.6.4 Get the synchronisation status

Use the following command to see the synchronization status:

```
ntpq -p
```

For more information, see http://nlug.ml1.co.uk/2012/01/ntpq-p-output/831.

2.6.5 Prepare to migrate the Echidna appliance to another hypervisor

Before migrating the virtual machine to another hypervisor, follow these steps:

1. Configure both hypervisors to use the same NTP server.
2. Ensure that the virtual machine is configured to update its clocks from only one of the following locations:
   - An external source (for example, Ubuntu's NTP server configured in the guest OS).
   - The host (for example through VMware Tools)

VMware recommends using NTP rather than VMware Tools periodic time synchronization. The VMware knowledge base also lists some configuration for Linux Guests.

Chapter 3: Configure the Echidna database

Echidna uses a database to store the following records:

- **Token records**: These are stored in Echidna's database if Echidna is handling authentication itself. Where Echidna brokers authentication requests to another authentication server, it does not store its own token records.
- **Audit and event logs**
- **User records:** (Optional) Echidna can store user records in its database. This can be useful when Echidna protects a small number of accounts. If there is an existing user store, there is no need to duplicate user records in a database.

Immediately after installation, Echidna uses an internal Derby database. Echidna can continue to use this internal database, or it can be configured to use an external database.

- The internal database is limited only by disk space on the appliance or application server.
- Use of an external database connection allows administrators to take better control of backup and maintenance on the database contents.

  In addition, an external database can be shared by two Echidna servers. This allows the two Echidna servers to share token records.

In either situation, consider creating a database recovery strategy so that no user or token records are lost. To preserve the whole Echidna configuration state (including the internal database contents), back up the entire contents of the Echidna configuration folder. For more information, see Backup and recovery on page 130.

3.1 Supported database types

The Echidna appliance comes pre-installed with the JDBC client libraries for the following database types:

- Derby
- MySQL
- Oracle
- Microsoft SQL

If the external database is not one of those, install the JDBC driver JAR file in the /usr/share/tomcat7/lib folder on the appliance.

3.2 Connect to an external database

1. Log in to the Administration console.
2. Click the **User Stores** tab.
3. In the **JDBC Connections** section on the left, click **new**.
4. Enter a name for the new connection, select the database type, then click **Next**.
   The new JDBC connection is created with default settings.
5. Update any of the following settings:

Parameter	Description
Bind Credential	The username and password used to connect to the database. • **Username:** The username for identifying the bind entity (either full DN or name@domain UPN format). • **Password**: The (protected) password for authenticating the bind entity. • **Password Protector**: The password protector that protects this password. Without a password protector, the password is stored in plain text. • **plaintext password recorded:** Indicates that a password has been configured, but no password protector was used and therefore a plaintext password has been stored. This indicates that a password has been set. • **Protected Value**: This displays the encrypted value of the password if a Password Protector has been set. • **New Password**: The password for authenticating the bind entity.
JDBC Driver Class	The JDBC driver class used to connect to the database.
JDBC Connection URL	The JDBC connection URL used to connect to the database.
Name	
Params	The list of generic key-value configuration pairs

Parameter	Description
DBCP Connection Pooling	• **Enabled**: Defines the apache-commons-DBCP connection pooling settings to use. • **Not enabled**: DBCP connection pooling is not used, and the JDBC driver is used directly to obtain a connection each time.
Maximum Active Connections	Controls the maximum number of objects that can be allocated by the pool (checked out to clients, or idle awaiting checkout) at a given time. When non-positive, there is no limit to the number of objects that can be managed by the pool at one time. When maxActive is reached, the pool is said to be exhausted. The default setting for this parameter is 8.
Maximum Idle Connections	Controls the maximum number of objects that can sit idle in the pool at any time. When negative, there is no limit to the number of objects that may be idle at one time. The default setting for this parameter is 8.
Minimum Idle Connections	The minimum number of objects allowed in the pool before the evictor thread (if active) spawns new objects.
Action When Pool Exhausted	Specifies the behaviour of the borrowObject() method when the pool is exhausted: • **FAIL**: borrowObject() will throw a NoSuchElementException • **BLOCK**: (Default) borrowObject() will block (invoke Object.wait()) until a new or idle object is available. If a positive maxWait value is supplied, then borrowObject() will block for at most that many milliseconds, after which a NoSuchElementException will be thrown. If maxWait is non-positive, the borrowObject() method will block indefinitely. The default whenExhaustedAction setting is WHEN_EXHAUSTED_BLOCK and the default maxWait setting is -1. By default, therefore, borrowObject will block indefinitely until an idle instance becomes available. • **GROW**: borrowObject() will create a new object and return it (essentially making maxActive meaningless.)
Maximum Wait Time (milliseconds)	
Test On Borrow	When testOnBorrow is set, the pool will attempt to validate each object when it is obtained from the pool (using the configured validation query). Objects that fail to validate will be dropped from the pool, and a different object will be borrowed. The default setting for this parameter is false.
Test On Return	When testOnReturn is set, the pool will attempt to validate each object before it is returned to the pool (using the configured validation query). Objects that fail to validate will be dropped from the pool. The default setting for this parameter is false.

Parameter	Description
Time Between Eviction Runs (milliseconds)	Indicates how long the eviction thread should sleep before "runs" of examining idle objects. When non-positive, no eviction thread will be launched. The default setting for this parameter is -1 (i.e., idle object eviction is disabled by default).
Minimum Evictable Idle Time (milliseconds)	Specifies the minimum amount of time that an object may sit idle in the pool before it is eligible for eviction due to idle time. When non-positive, no object will be dropped from the pool due to idle time alone. This setting has no effect unless timeBetweenEvictionRunsMillis > 0. The default setting for this parameter is 30 minutes.
Number of Tests Per Eviction Run	Determines the number of objects examined in each run of the idle object evictor. This setting has no effect unless timeBetweenEvictionRunsMillis > 0. The default setting for this parameter is 3.
Test While Idle	Indicates whether or not idle objects should be validated using the configured validation query. Objects that fail to validate will be dropped from the pool. This setting has no effect unless timeBetweenEvictionRunsMillis > 0. The default setting for this parameter is false.
Validation Query	A query to use to validate connections. Should return at least one row. Using null turns off validation.
Default Read Only	The default "read only" setting for borrowed connections.
Default Auto-Commit	The default "auto commit" setting for returned connections.
Pool Prepared Statements	Whether to use a prepared-statement pool factory. If true, a StackKeyedObjectPoolFactory with maximum size 8 and initial size 0 is used for prepared statement pooling.

6. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
7. (Optional) Test the JDBC connection.

   Warning: Testing the connection removes all unsaved changes from the page. Save the configuration before testing the connection.
8. Consider backing up the new configuration as a file. See Backup and recovery on page 130.

3.3 Manage the Echidna database

The internal database can be exposed to external JDBC-based tools for management by using the Derby Network Server listener service. To use this service, install the **derbynet-10.8.2.2.jar** library in the /usr/share/tomcat/lib folder on the appliance.

Chapter 4: Configure the user data

This chapter describes how to connect Echidna to one or more user stores. It covers the part of Echidna configuration that is shown in the following diagram.

This diagram shows the order in which the administrator configures the user data, starting with the **user resolution process**.

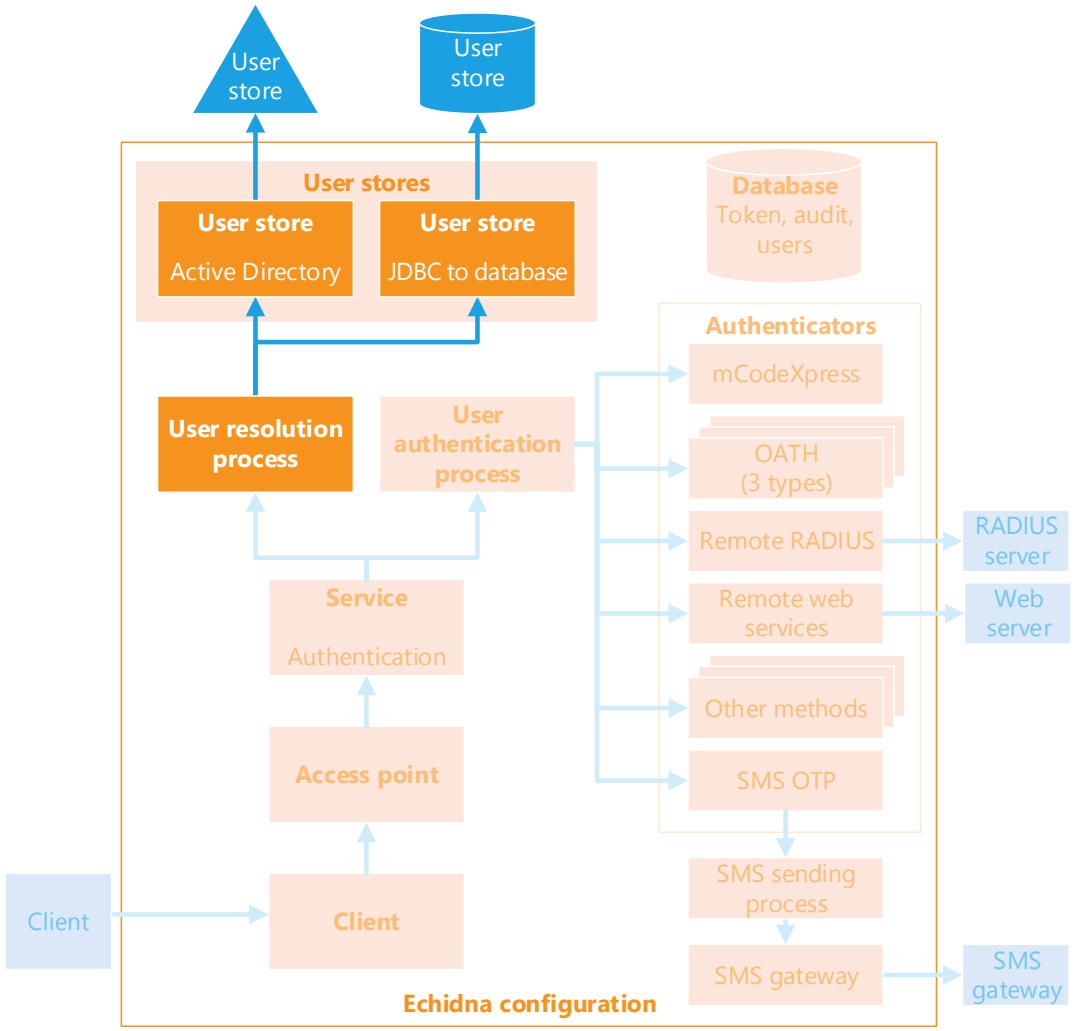


Figure 5: The configuration that deals with user data

4.1 How Echidna uses user data

Echidna authenticates tokens and users. When authenticating users, Echidna needs an authoritative source of data to authenticate against: a user store.

A user store is a source of information about the users that Echidna authenticates. A user store includes information such as each user's name, password, phone number, and group memberships. Each user record can be associated with token serial numbers.

If an organisation already uses a directory or database as a user store, connect Echidna to this existing store. If an organisation has no external user store, configure Echidna to create its own internal user store.

Echidna can connect to more than one user store. This may be multiple Active Directory domains, or may represent different user groups such as staff and external contractors. Normally, Echidna resolves users based on the domain part of the user ID, but this can be changed to search multiple stores, if required.

This table describes the types of user store that Echidna can use:

Name	Description
Active Directory LDAP	An Active Directory store
Generic LDAP Directory	Any other LDAP directory
JDBC with Existing Table	An existing table in a database. Use this option if the organisation already uses a database as a user store.
JDBC with New Table	A new table in a database. Use this option if the organisation maintains a database, but does not already use it as a user store. Echidna defines and creates its own user and group membership tables.
Internal	A limited number of user and group records stored in Echidna's configuration file. These internal users and groups let administrators test and develop a user data model before migrating those records into an external store.
Implicit	A virtual user store This allows Echidna to provide authentication services without reference to a user store with the actual user records. This can support proxying to an external authentication service where all users not otherwise found by Echidna are assumed to be known by the external server.

4.2 Link to a user store

This section describes how to link Echidna to a user store.

1. Log in to the Administration console.
2. Click the **User Stores** tab.
3. Under **User Stores** on the left, click **new**.
4. Link to the user store:
   - **New userstore name**: Enter the name for the new user store. Echidna treats this user store name as a domain name when disambiguating multiple user stores.
   - **Select userstore type**: Choose Active Directory LDAP or Generic LDAP Directory.
   - **Select the existing User-Resolve processes that should reference the new store**: For each check mark, the new store is added to the resolve user process. The store name is used as the domain name. If it is not checked, it can be added to a user resolution process later. For more information, see How processes work on page 135.
5. Click **Next**. The user store is immediately added to the Echidna configuration.
6. Fill out the additional configuration for the type of user store selected:
   - LDAP directory (see page 26).
   - JDBC database (see page 29).
   - Internal user store (see page 31).
   - Implicit user store (see page 32).
7. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
8. Consider backing up the new configuration as a file. See Backup and recovery on page 130.

4.2.1 Linking to an Active Directory or other LDAP user store table

This section applies to both of the LDAP user store options.

The configuration required for these two types of user store is identical. However, the default values for the base DN and search filters are different.

LDAP Parameter	Description
Domain Name	The name of the user domain for this store, if it is different from the store name itself.
Description	A description of this store suitable for display in the User Support console.
User ID Filter Pattern	If given, check any submitted user ID values against this regex pattern before attempting to resolve the user in the store.
Password Validity Filter	If given, check any submitted password values against the given character set before attempting to validate in the user store.
Username	The user name for binding to the user store. This can be either a full DN or for Active Directory it can be in the UPN format **name@domain**.
Password Protector	The password protector that protects this password.
Protected value	This displays the encrypted value of the password if a Password Protector has been set.
New Password	The password for authenticating the bind entity.
Service URL	The URL for accessing this LDAP user store.
SSL Context	An SSL Context defines a password-protector (a key in a keystore) and a trust store that Echidna can use for SSL or TLS connections. The default context is **sslctx**. This context points to the Echidna **serverIdentity** key and the **cacerts** truststore belonging to the JDK that Echidna is using.
Use StartTLS	Check this box to attempt to use StartTLS with this LDAP server. Some LDAP servers are able to use a single port for binding both **ldap** and **ldaps**, but then to negotiate to use TLS over that connection using the **StartTLS** operation.
JNDI Context Environment	Details of the connection to the LDAP user store. The most common options (for handling referrals and connection pooling) are pre-populated. See the Oracle JNDI/LDAP documentation for supported properties at http://docs.oracle.com/javase/7/docs/technotes/guides/jndi/jndi-ldap.html
User Search, Search Base DN	The base DN from which to search for the users in the user store.

LDAP Parameter	Description
User Search, LDAP Search Filter	The search filter that identifies users. Default values: • **Active Directory**: (&(objectClass=person)(sAMAccountName={0})) • **Generic LDAP Directory**: (&(objectClass=person)(uid={0}))
User Search, Search Scope	The part of the directory tree that Echidna searches for users.
User Search, Search Filter for Listing	The search filter for listing all user objects in the store. This is normally the same as the user search filter with the user ID term removed.
Role Search, Search Base DN	The base DN from which to search for groups in the user store.
Role Search, LDAP Search Filter	The search filter that identifies roles. Default values: • **Active Directory**: **(&(objectClass=group)(cn={1})(member={0}))** • **Generic LDAP Directory**: **(&(objectClass=groupOfUniqueNames)(cn={1})(uniqueMember={0}))**
Role Search, Search Scope	The part of the directory tree that Echidna searches for roles.
Role Search, Search Filter for Listing	The search filter for list all group objects in the store. Normally the same as the above role search filter with the group name and user ID terms removed.
Trace, Label	Label of tracer to show in the User Support console.
Trace, History Buffer Size	Specifies the number of trace records to cache for viewing. Trace records are viewable via the Monitoring/Call Trace option and via the User Support console.
Trace, Current State Summary	These settings are used to track the summary of the number of successful and failed LDAP invocations over recent time windows. The summary records are viewable via the Monitoring/Traffic page. If enabled, summary statistics are kept about the recent events in Bucket Count buckets, each representing a **Bucket Duration** period of time. For example, Bucket Duration=120 seconds, Bucket Count=112, Time Step=10.0 show the stats for the last 2 minutes, the last 20 minutes before that, and the last 200 minutes before that.
Trace, Trace Log	Settings for recording traces of outgoing LDAP invocations to a flat file.
Trace Log, Trace Log Path	The path to the log file, relative to the configuration XML file.
Trace Log, Trace Entry Format	Leave empty for default log-entry format.
Trace Log, Rollover Limit (Bytes)	Approximate maximum amount to write (in bytes) to any one file. If zero, then there is no limit.
Trace Log, File Count	How many output files to cycle through. Zero means keep creating new files.

LDAP Parameter	Description
Trace Log, Log Level	One of the following logging levels: • OFF (or blank) • SEVERE • WARNING • INFO • CONFIG • FINE • FINER • FINEST • ALL
Trace Log, Log Message Escaping	The type of escaping to apply to the messages before formatting them to the log output.
Trace Log, Stack Trace Maximum Lines	The maximum number of lines of the stack-trace to include in the logs. Possible options: • 0: No limit • -1: Do not include the stack trace in the log files
Trace, Persistent Log	Settings for recording outgoing LDAP invocation events to a database.
Persistent Log, Datastore Connection Name	The database that is used to record LDAP invocations.
Attribute Aliases	Sometimes an authentication method might need to refer to common attributes across multiple user stores to query or store information to. Aliases are a user-friendly way to reference existing database or directory attributes. For example, an Active Directory user has an attribute named **carLicense**, which is used to hold a token ID for a particular authenticator. An alias could be created for **carLicense** called **tokenid**.
Attribute Triggers	Creates an attribute update process, which is rarely used.
User Store Lookup	Use the **Find User**, **Validate Password** and **List Users** buttons to test this user store configuration by looking up user records or testing the password validation.

4.2.2 Linking to a JDBC database with or without an existing table

Edit any of the following parameters:

Database Parameter	Description
Domain Name	The name of the user domain for this store, if it is different from the user store name.
Description	A description of this user store. This description appears in the User Support console.
User ID Filter Pattern	If given, check any submitted user ID values against this regex pattern before attempting to resolve the user in the store.
Password Validity Filter	If given, check any submitted password values against the given character set before attempting to validate in the user store.
Username Attribute	The database column that holds the user name.
Display Name Attribute	The database column that holds the display name.
Password Attribute	The database column that holds the password.
Created When Att	The database column that holds the time that the user record was created.
Expires When Att	The database column that holds the time that the user record will expire.
Status Att	The database column that holds the status of the user record. The user table schema may be defined by an external application, which means that any value is possible. However, a common practice is to use an integer value where 0 means enabled and any other values give reason codes for why the user record is disabled. Named values (enabled, disabled, suspended) are also possible.
Status Available Pattern	The pattern that identifies which user records are available. This is used only if Status Att is defined. If both of these are defined, a user record is available for authentication if its Status Att matches this pattern. This is a regular expression. If it is left blank, the default is to accept a status value of 0 as meaning available.
Other Attributes	This is not required.
Persistence	This section defines the database connection and relevant user and group tables along with the update and query statements for those tables. At a minimum there should be **users.fetch** and **users.list** named queries defined, and a table definition for the table referenced in those queries. Each column of the main user table (default ECHIDNASTORE_USERS) defines a single-valued user attribute. Additional tables with foreign-key constraints referencing the main user table define multi-valued user attributes (such as group membership).

Database Parameter	Description
Attribute Aliases	Sometimes an authentication method might need to refer to common attributes across multiple user stores to query or store information. Aliases are a user-friendly way to reference existing database or directory attributes. For example, an Active Directory user has an attribute named **carLicense**, which is used to hold a token ID for a particular authenticator. An alias could be created for **carLicense** called **tokenid**.
Attribute Triggers	Creates an attribute update process, which is rarely used.
User Store Lookup	Use the Find User, Validate Password and List Users buttons to test the user store by looking up user records or testing the password validation.

4.2.3 Linking to an internal user store

When an administrator uses the Echidna internal store type as a user store, the user store configuration directly contains the user records and role membership records.

Internal DB Parameter	Description
Domain Name	The name of the user domain for this store, if it is different from the store name itself.
Description	A description of this store suitable for display in the User Support console.
User ID Filter Pattern	If given, check any submitted user ID values against this regex pattern before attempting to resolve the user in the store.
Password Validity Filter	If given, check any submitted password values against the given character set before attempting to validate in the user store.
Users	The users in the internal user store. Use the **Add** button to add users.
User Name	The user name (**userid**) for this user record.
User Password	The password for authenticating this user record.
Password Protector	The password protector that protects this password. Without a password protector, the password is stored in plain text.
plaintext password recorded	Indicates that a password has been configured, but no password protector was used and therefore a plaintext password has been stored. This indicates that a password has been set.
Protected Value	This displays the encrypted value of the password if a Password Protector has been set.
New Password	The password for authenticating the bind entity.
User Attributes	The list of generic attribute key and value pairs. These become the defined user attributes in the resolved user context. For example, specify key="mobile" and value="5551234" so that the user context has a mobile value defined.
Roles	The roles (groups) in the internal user store. Use the **Add** button to add roles.
Member	The users who are members of this role. Enter names of users in this internal user store.
Attribute Aliases	Sometimes an authentication method might need to refer to common attributes across multiple user stores to query or store information to. Aliases are a user-friendly way to reference existing database or directory attributes. For example, an Active Directory user has an attribute named **carLicense**, which is used to hold a token ID for a particular authenticator. An alias could be created for **carLicense** called **tokenid**.

Internal DB Parameter	Description
Attribute Triggers	Creates an attribute update process, which is rarely used.
internal User Store Lookup	Use the **Find User**, **Validate Password** and **List Users** buttons to test this user store configuration by looking up user records or testing the password validation.

4.2.4 Linking to an implicit user store

This user store is a virtual user store.

This allows Echidna to provide authentication services without reference to a user store with the actual user records. This can support proxying to an external authentication service where all users not otherwise found by Echidna are just assumed to be known by the external server.

Implicit user store parameter	Description
Domain Name	The name of the user domain for this store, if it is different from the store name itself.
Description	A description of this store suitable for display in the User Support console.
User ID Filter Pattern	If given, check any submitted user ID values against this regex pattern before attempting to resolve the user in the store.
Password Validity Filter	If given, check any submitted password values against the given character set before attempting to validate in the user store.
Attribute Aliases	Sometimes an authentication method might need to refer to common attributes across multiple user stores to query or store information to. Aliases are a user-friendly way to reference existing database or directory attributes. For example, an Active Directory user has an attribute named **carLicense**, which is used to hold a token ID for a particular authenticator. An alias could be created for **carLicense** called **tokenid**.
Attribute Triggers	Creates an attribute update process, which is rarely used.
implicit User Store Lookup	Use the **Find User**, **Validate Password** and **List Users** buttons to test this user store configuration by looking up user records or testing the password validation.

4.3 Configure a user resolution process

Echidna administrators can configure the way that target users are resolved from the user stores. Echidna extracts the user name and domain from the User ID field during login, using regular expressions. Echidna then decides which user store to search for the user.

This means that after user store is added to or removed from Echidna, the user resolution process must be updated to reflect the change.

1. Log in to the Administration console.
2. Click the **User Stores** tab.
3. Under **User Resolution Procs** on the left, click **new**.
4. Enter a name for the new user resolution process.
5. Select some or all of the user/domain patterns to recognize.
6. Select some or all of the existing user stores. If a user store is created later, it can be added to the process then.
7. Click **Next**.
8. Define the conditions.
9. Use the **Add**  , **Delete**  , **Move Up**  , and **Move Down**  buttons to change the conditions if necessary.

   For information about these conditions, see How processes work on page 135.
10. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
11. Consider backing up the new configuration as a file. See Backup and recovery on page 130.

Salt recommends that administrators limit their changes to these conditions to updating the domain-matching expressions. For example, to allow the same user store to be known as either **xy** or as **xy.ap.corp.net**, change from **CASEIGNOREMATCH** on **hk.ap.corp.net** to REGEX on **(?i)hk(\.ap\.corp\.net)?**.

4.4 Make Echidna search multiple stores

The default **user resolution process** contains the following conditions:

1. A **choose** condition to extract the username and domain from the request. User-Name with sub-conditions to match the supported patterns (username@domain, domain\username, default domain).
2. A **choose** condition to search for the user in the appropriate user store based on the domain extracted in the first step, with sub-conditions for each store.

To search multiple stores, the second **choose** condition can be replaced with an **always** condition to search the first store, and then a series of **not-present** conditions to keep searching the subsequent stores until the **userContext** variable becomes defined.

4.5 Access to the User Support and Self Service consoles

Echidna comes with two supporting consoles:

- **User Support console**: Used by support staff to manage end users and their tokens.
  By default, **no** users have access. This protects Echidna until it is ready for deployment.
- **Self Service console**: Used by end users to register and manage their own tokens.
  By default, **all** users in the user store have access.

The URLS for these consoles are listed on the appliance console screen.

4.5.1 List of default roles

After installation, Echidna includes default roles that grant access to the two consoles. The following roles grant various levels of access to the **User Support console**:

Role Name	Label	Description	Default Membership Rule
user-support-reader	Readonly User Admin	Members can sign in to the User Support console but not make any changes. This role is useful for auditors.	Must be a member of the **GRANT 2FA-User Readonly Administration** group.
user-support-updater	Full User Admin	Members can sign in to the User Support console and make changes to the tokens and users.	Must be a member of the **GRANT 2FA-User Full Administration** group.
token-admin	Token Administrator	Members can sign in to the User Support console to import and manage tokens only.	Must be a member of the **GRANT 2FA-Token Administration** group.
userreg-admin	User Registrar	Members can sign in to the User Support console to create, edit, and delete user records in a database store. By default, the service that permits user editing is not enabled. For steps to enable this service, see Allow users and groups to be managed in the User Support console on page 37.	Must be a member of the **GRANT 2FA-User Registration** group.

The following role grants access to the **Self Service console**:

Role Name	Label	Description	Default Membership Rule
remote-user	Remote Access User	Defines which users should be permitted to perform two-factor authentication for remote access.	No restriction. Any user that the resolve process can identify is a valid remote user.

For each role, the membership rules can be edited to include a list of groups and a list of attributes with particular patterns that the resolved user entry must satisfy.

Additional roles can be defined as needed. Contact the Echidna supplier for advice about creating custom roles.

4.5.2 Give access to the User Support console

Access to the User Support console is controlled by roles stored in Echidna. For a full list of these roles and the permissions they grant, see List of permissions for the default User Support console roles on page 40.

To grant access, update the membership rules for the roles in Echidna or assign the users to the relevant user store groups. Roles in Echidna can be granted by group membership or by the presence of named attributes with matching values in the user store entries.

By default, each role is restricted to those users who are members of a particular group. See the rightmost column in the table in List of default roles on page 34 for the names of the groups. A user store is very unlikely to contain groups with those names, which effectively means that no-one can sign in to the User Support console. This protects Echidna until it is ready for use.

To allow support staff to sign in to the User Support console, either update the membership rule in the Echidna configuration to include groups that actually exist, or create or rename groups in the user store to match the default rules. The following steps describe how to update the Echidna configuration to match groups that already exist in the user store.

1. In the user store, identify or create a group that contains only users that should be allowed to manage users and tokens in the User Support console.
2. Log in to the Administration console.
3. Click **CONFIG**, then click **Authenticators**.
4. In the list on the left, select **AuthenticationProc**.
5. Scroll down to the **User Roles** section, then find the following sections:
   - **user-support-updater**: Defines the groups with full access to the User Support console
   - **user-support-reader**: Defines the groups with read-only access to the User Support console
6. In the **Granting Groups** section, change **Group Name** to match the actual group in the user store.
7. If required, use the **Add** button  on the far right to add more groups.
8. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
9. Consider backing up the new configuration as a file. See Backup and recovery on page 130.

4.5.3 Restrict access to the Self Service console

Echidna includes a **Remote Users** role, which controls access to the **Self Service** console. All users in this role can access the Self Service console. See List of permissions for the remote-user role on page 42 for the list of permissions that are granted by this role.

After the Setup wizard is complete, the **Remote Users** role includes all users from all user stores as members. This lets all users sign in to the Self Service console and set their own login method preference and register their own tokens.

If this works well in an organisation, there is no need to change the Echidna configuration. However, it is possible to restrict the Self Service console to only some users.

Use these steps to allow only some users to access the Self Service console.

1. Log in to the Administration console.
2. Click **CONFIG**, and then click **Authenticators**.
3. In the list on the left, select **authenticationProc**.
4. Scroll down to the **User Roles** section, and then find the **remote-user** section. This section defines the **Remote Users** role.
5. To restrict access to only these users with a particular *attribute*:
   a. Choose an option in the **Authentication Attributes Combination** list:
      - **ONE_OF**: Users with at least one of the following attributes will be able to sign in to the Self Service console
      - **ALL_OF**: Only users with all of the following attributes will be able sign in to the Self Service console.

   b. Add an attribute by clicking the **Add** button on the far right of the **Auth Attributes** table. Use the **Delete** button to remove an attribute.
6. To restrict access to only these users in a particular *group*:
   a. Choose an option in the **Granting Groups Combination** list:
      - **ONE_OF**: Users in at least one of the following groups will be able to sign in to the Self Service console. If the list is empty, no users will have access.
      - **ALL_OF**: Only users in all of the following groups will be able sign in to the Self Service console. If the list is empty, all users will have access.

   b. Add a group by clicking the **Add** button on the far right of the **Granting Groups** table. Use the **Delete** button to remove a group.
7. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
8. Consider backing up the new configuration as a file. See Backup and recovery on page 130.

4.5.4 Allow users and groups to be managed in the User Support console

By default, Echidna does not allow users and groups to be edited in the User Support console.

In many organisations, this works well, because they already have tools for editing users. However, if an organisation needs the User Support console to let support staff edit users, this can be turned on. It works only for users that are stored in a database, not in an LDAP directory or Active Directory.

The following steps describe how to expand the User Support console to allow its users to edit, create, and delete users and groups.

1. Log in to the Administration console.
2. Click **CONFIG**, then click **Connectors**.
3. In the **Services** section on the left, click **webServices**.
4. Add a new service for the extra functionality:
   a. Scroll down to the **User Reg Services** section.
   b. Click the green **Add** button  .
   c. Type **userReg** in the **New Name** box, then click **Add**.

   The new **userReg** service is now listed in both the **User Reg Services** section, and in the **Published Services** section.
5. In the **Published Services** list, check the **Service Enabled** box for the **userReg** service.
6. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
7. Consider backing up the new configuration as a file. See Backup and recovery on page 130.

4.5.5 Enable the Self Service console

If the administrator used the Setup wizard, the Self Service and User Support consoles work immediately.

If the configuration has been changed, or if the administrator did not use the Setup wizard, the application requires extra configuration. Use the information in this section to enable the Self Service console.

The Self Service and User Support consoles use Echidna web services on the same host. Ensure that these web services are available. Also, ensure that the following elements are also available:

1. Log in to the Administration console.
2. Click the **Setup** tab.
3. Configure the localhost Echidna client:
   a. Click step **6. Clients**.
   b. Click **Set Random**. This creates a shared secret for localhost.
   c. Click **Next** to save this shared secret in **localhost.cred**. This new file is saved in the configuration folder.
4. Configure an HTTP web service listener:
   a. Click step **7. Service**.
   b. Check the values in the **Web Service Listener** section. The default port for this listener is 1888.
   c. Click **Next** to save these values.
5. Click the **Connectors** tab.
6. In the **Services** list on the left, select **WebServices**.
7. Ensure that the following authentication services are available. To do this, find the **Published Services** section, then check the **Service Enabled** box for each service listed here
   - **intUserAuth**: Defines a one-factor (username/password) authentication process. This service allows login to the applications. This service was defined automatically in step **7. Service** in the Setup wizard.
   - **webUserAuth**: Defines which users and which authentication methods are managed through the support pages. This service was defined automatically in step **7. Service** in the Setup wizard.
   - **traceLogs**: Queries the authentication event records and database audit tables.
   - **Services that are related to authentication methods (such as mCodeXpressAuth)**

4.5.6 Design other roles

Administrators can define additional roles that can be used by the User Support and Self Service consoles to grant access to pages and actions in those applications.

Any roles defined here can be used within the authentication processes using the **AVAILABLE** conditional term. The default authentication process checks if the remote-user role is **NOT AVAILABLE**, and returns a **USER_UNAUTHORIZED_RES** result if that is the case.

To add other roles:

1. Log in to the Administration console.
2. Click the **Authenticators** tab.
3. In the list on the left, select **AuthenticationProc**.
4. Click the **Add** button  next to **User Roles**.
5. Enter the name of the role in **New External Name**, then click **Add**.
6. Find the new role, then update its label, description, and group membership requirements.
7. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
8. Consider backing up the new configuration as a file. See Backup and recovery on page 130.

New roles can also be created by editing the allowed-role definitions in the user-support-config.xml and self-service-config.xml file. Contact the Echidna supplier for advice about creating custom roles.

4.5.7 List of permissions for the default User Support console roles

The User Support console lets users view and manage users and tokens.

This table lists the four roles that are supplied with Echidna, and each role's permission for each task in the User Support console.

Task in the User Support console	Description	Read-only Operator	Full Access Operator	Token Administrator	User Management
Lookup by User	Operator can update editable attributes and set the user's login method preference	No	Yes	No	No
	Operator can view the current state	Yes	Yes	No	No
Trace Info	Operator can query the audit records	Yes	Yes	No	No
Activate mCodeXpress	Operator can activate (register) mCodeXpress tokens	No	Yes	No	No
	Operator can cancel mCodeXpress registration (the token deleted)	No	Yes	No	No
Activate OATH Token	Operator can assign OATH tokens	No	Yes	Yes	No
Activate YubiKey OATH Token	Operator can activate YubiKey OATH tokens.	No	Yes	No	No
Activate Temporary Password	Operator can assign temporary (single-use) access passwords	No	Yes	No	No
Activate SMS OTP	Operator can assign/register an SMS OTP	No	Yes	No	No
Manage mCodeXpress Tokens	Operator can view the current state of the token	Yes	Yes	Yes	No
	Operator can cancel the mCodeXpress registration (the token deleted)	No	Yes	Yes	No
	Operator can revoke the mCodeXpress token	No	Yes	Yes	No
	Operator can suspend the mCodeXpress token	No	Yes	Yes	No
	Operator can un-suspend the mCodeXpress token	No	Yes	Yes	No
	Operator can re-sync the mCodeXpress token	No	Yes	Yes	No
	Operator can validate the mCodeXpress token's OTP				
	Operator can look up user by token ID	Yes	Yes	No	No

Task in the User Support console	Description	Read-only Operator	Full Access Operator	Token Administrator	User Management
Manage OATH Tokens	Operator can view the current state of the Token	Yes	Yes	Yes	No
	Operator can cancel the OATH Token registration (the token deleted)	No	Yes	Yes	No
	Operator can revoke the OATH token	No	Yes	Yes	No
	Operator can suspend the OATH token	No	Yes	Yes	No
	Operator can un-suspend the OATH token	No	Yes	Yes	No
	Operator can re-sync the OATH token	No	Yes	Yes	No
	Import OATH Token Seeds	No	No	Yes	No
	Operator can look up user by token ID	Yes	Yes	No	No
Manage YubiKey Tokens	Operator can view the current state of the Token	Yes	Yes	Yes	No
	Operator can cancel the YubiKey Token registration (the token deleted)	No	Yes	Yes	No
	Operator can revoke the YubiKey token	No	Yes	Yes	No
	Operator can suspend the YubiKey token	No	Yes	Yes	No
	Operator can un-suspend the YubiKey token	No	Yes	Yes	No
	Operator can re-sync the YubiKey token	No	Yes	Yes	No
	Operator can look up user by token ID	Yes	Yes	No	No
Manage SMS OTP	Operator can view the current state of the Token	Yes	Yes	Yes	No
	Operator can deleted the SMS OTP Token registration (the token deleted)	No	Yes	Yes	No
	Operator can revoke the SMS OTP token	No	Yes	Yes	No
	Operator can suspend the SMS OTP token	No	Yes	Yes	No
	Operator can un-suspend the SMS OTP token	No	Yes	Yes	No
	Operator can re-sync the SMS OTP token	No	Yes	Yes	No
	Operator can look up user by token ID	Yes	Yes	No	No
Manage Users	The operator can view, create, update and delete users and groups in a configured editable users store.	No	No	No	Yes

4.5.8 List of permissions for the remote-user role

The Self Service console has the following possible tasks:

- User can set their own login method preference
- User can view the current state of their token
- User can register mCodeXpress tokens
- User can activate YubiKey OATH token
- User can view the current state of their mCodeXpress token
- User can validate the mCodeXpress token's OTP
- User can view the current state of their OATH token
- User can view the current state of the Yubikey token
- User can re-register the YubiKey token (so separate re-sync is not required)
- User can validate the YubiKey token's OTP
- User can view the current state of an SMS OTP token

All tasks in the Self Service console are available to all users who can log in. When an authentication mechanism (such as Salt mCodeXpress) is enabled for Self Service, all tasks are available.

Chapter 5: Configure authentication

This chapter describes how to set up authentication. It covers the many types of authenticators, and the user authentication process, as shown in this diagram:

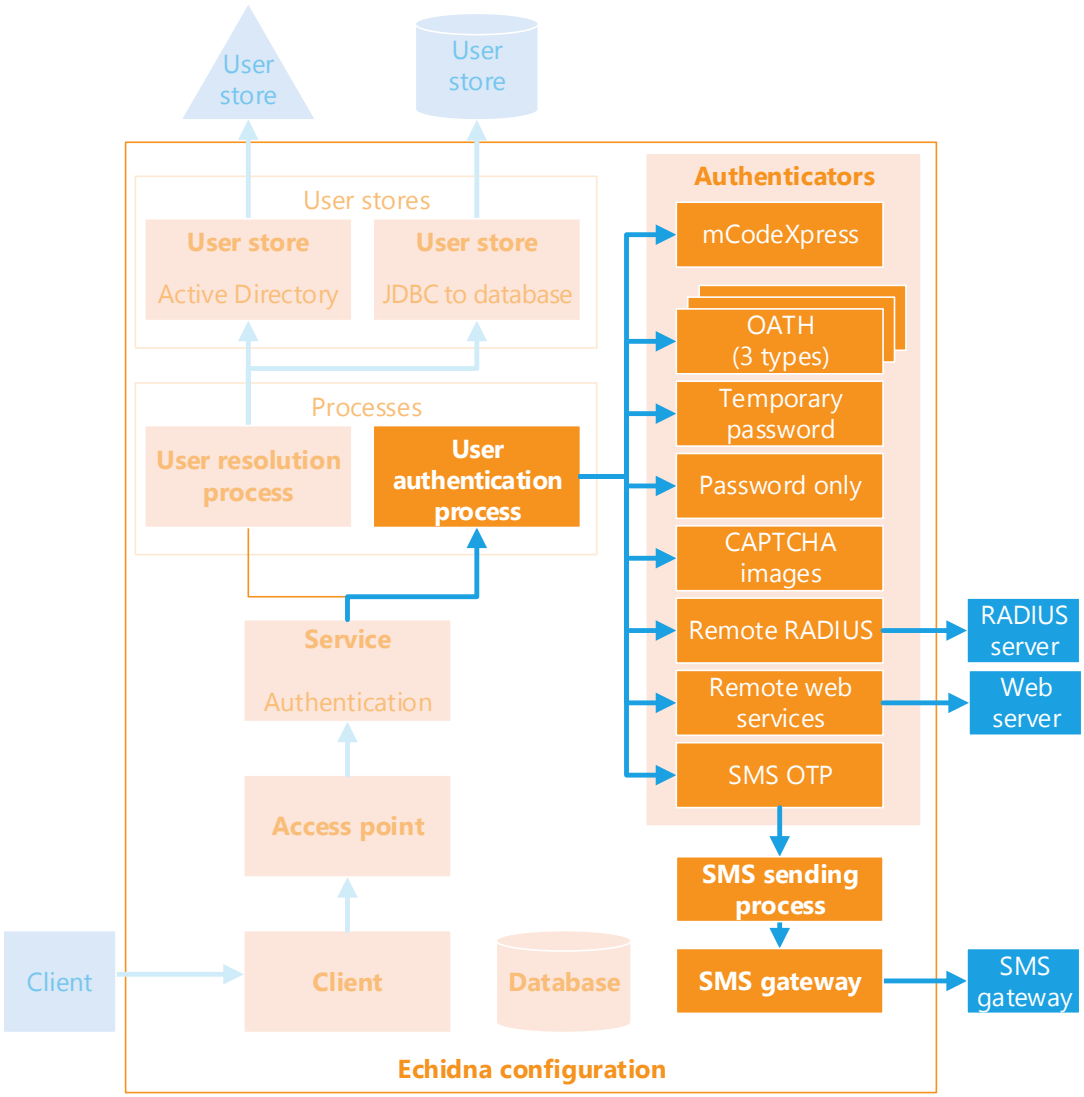


Figure 6: The configuration that deals with authentication

5.1 How authentication works in Echidna

To achieve two-factor authentication, a user's knowledge of some secret (password or PIN) is usually combined with a user's possession of some device (mobile phone, security token).

Some security tokens (like one-button OATH key-fob tokens) have no built-in PIN or password associated with them. In that case, it may be desirable to combine the OTPs produced by the tokens with a validation of a password from the user store.

Echidna lets administrators set up as many authenticators as they need. Each authenticator contains the configuration for a single authentication method.

In addition, administrators can set up many authentication processes. Each of these processes defines how authentication requests are handled, including the following:

- **Define which authentication methods are available:** The authenticator is usually chosen based on the attributes available for a user.

  For example, a user can use Salt mCodeXpress only if the user has a Salt mCodeXpress token identifier in the user store.
- **Link groups to tokens**: For example, a user can use SMS if their record contains a phone number AND they are in a particular group.
- **Check for user lockouts**.

Together, authenticators and authentication processes let administrators configure how to authenticate users who have different sets of attributes from one another, as in multi-domain environments.

Support for each authentication method includes direct use of authentication or transaction signing, as well as full lifecycle management as it applies to each method – token import, registration, assignment, activation, suspension and revocation.

5.2 Types of authenticator

This section lists the authenticators that Echidna supports. Each authenticator uses a single authentication method.

Authenticator	Type	Description
Salt mCodeXpress OTP (page 51)	Token	Handles OTPs that were created by the Salt mCodeXpress app, which is supplied by Salt Group.
SMS-delivered OTP (page 54)	Token	Handles OTPs that were sent to the mobile phone number that is recorded in the user store.
OATH HOTP (Generic) (page 58)	Token	Handles OATH event-based OTP tokens, which are supplied by any token that supports the OATH HOTP standard (RFC 4226).
OATH TOTP (page 60)	Token	Handles OATH time-based OTP tokens, which are supplied by any token that supports the OATH TOTP standard (RFC 6238).
OATH OCRA (page 64)	Token	Handles OATH time- or event-based OTPs and signatures (RFC 6287).
OATH HOTP (YubiKey) (page 67)	Token	Handles YubiKey hardware tokens that are programmed as OATH event-based OTP generators.
Limited-use temporary password (page 70)	Static	Handles temporary passwords for emergency use, such as when the main authentication method is unavailable (perhaps due to a lost token or phone). These passwords expire after a set number of uses or a set time. These are delivered over the phone by the organisation's user support staff.
Password-only (page 73)	Static	Authenticates against the user-store password only.
CAPTCHA images (page 74)	Supplementary	Handles passwords from generated images, used to stop robot logins.
Remote RADIUS (page 86)	Proxy	Uses a RADIUS server to validate the OTP. Supports legacy tokens such as RSA SecurID and Vasco tokens.
Remote web services (page 89)	Proxy	Uses a web-services-based authentication server to validate the OTP.

5.3 Choosing an authentication method

Echidna enables multiple alternative authentication methods. Each end-user can use one or more of the authentication methods that are configured in Echidna. Selection of the actual method to use can follow one of the strategies below.

5.3.1 User nominated at login

Some authentication channels (such as form-based login pages) allow the user to choose an authentication method from a list. As long as the chosen authentication method is available for the given user ID, authentication can proceed.

5.3.2 User nominated before login

Many authentication channels (such as RADIUS) only allow for submission of user ID and password/passcode credentials. Echidna decides how to interpret the credential values that the user submits. One approach is to record a value in a preferred-method attribute of the user store. That way multiple methods can be available, but an administrator or the users themselves can update the preferred-method attribute to indicate which one should be active on current log-ins.

5.3.3 Automatic on priority order

Where there is no preferred-method attribute, Echidna makes a selection based on a priority order. For example, if a user is registered for mCodeXpress, that is used. Otherwise SMS OTPs are used.

In Echidna, an authentication method is available if both of the following conditions are true:

- The fundamental authentication method requirements are met – such as a recorded mobile phone number for SMS, or a linked token record for OATH tokens.
- The permission requirements are met. Echidna can restrict availability of a method to users with appropriate group memberships or attribute value patterns.

To change the priority of an authentication method:

1. Log in to the Administration console.
2. Click the **Authenticators** tab.
3. In the list on the left, select **AuthenticationProc**.
   This page lists all of the authentication methods that are enabled.
4. Use the **Move up** and **Move down** buttons to change the order of the auth methods.
5. Click **Update** to save the configuration changes.

5.4 Authentication processes

After the Setup wizard, Echidna is configured with two sample authentication processes:

- **authenticationProc**: This process is configured for two-factor authentication, which selects the Salt mCodeXpress authenticator by default. Any new authenticators except Password Only and CAPTCHA should be added to this process.
- **singleFactorAuthProc**: This process is configured for one-factor authentication, which selects the Password Only authenticator by default. If the CAPTCHA authenticator is configured, it should use this process.

Administrators can modify these processes or delete them and create new ones.

5.5 Combining Echidna authentication methods with user store passwords

Each Echidna authentication method provides an OTP (one-time passcode). Each OTP can be combined with user store passwords as follows:

- **PASSCODE_ONLY**: The user store password is not used, only the OTP.
- **PASSCODE_WITH_PASSWORD** and **PASSWORD_WITH_PASSCODE**: The user store password and the OTP are combined in a single field when submitted by the user.
- **PASSWORD_THEN_PASSCODE**: The user store password is collected first, then the OTP is challenged for as an additional input. For this option, the collected store password can either be validated immediately (before challenging for the OTP), or it can be securely held to be validated only after the OTP validity is confirmed.

Depending on which of the above combinations are required, the login flow that the user experiences is one of the following:

- The user is prompted for user ID and OTP only. Either the credentials are correct and login completes successfully, or they are wrong and the login is rejected.
- The user is prompted for user ID, password AND OTP. All three elements must be correct, but they are either accepted or rejected in a single step.
- The user is prompted for user ID and password. If the user ID is valid and authorized, the user is asked for an OTP from a particular authentication method. At each step after the initial user ID/password collection, the login can be accepted or rejected, or the user prompted for further information.

Echidna supports all of the above combinations, but some authentication methods (such as SMS) and some authentication clients restrict which options can be used.

5.6 Add an authenticator

1. Log in to the Administration console.
2. Click **CONFIG**, and then click the **Authenticators** tab.
3. In the **Authenticators** section on the left, click **new**.
4. Enter a name for the new authenticator and select the type of authenticator. Use the descriptions on the screen and later in this chapter.
5. Select the existing **Authentication Processes** that should reference the new authenticator or create a new authentication process later (see Add an authentication process on page 49).
6. In the **Action** section, click **Next**.

   The new authenticator is created with default values.
7. Change some or all of the default values. Use the descriptions on the screen and later in this chapter.
8. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
9. Consider backing up the new configuration as a file. See Backup and recovery on page 130.
10. If required, update the authentication process to handle the new authenticator.

5.7 Check the name of an authenticator

1. Log in to the Administration console.
2. Click **CONFIG**, and then click the **Authenticators** tab.
3. In the **Authenticators** section on the left, click the name of the authenticator.
4. Look in the **External Name** box. This is the name that Echidna uses to refer to this authenticator.

5.8 Add an authentication process

To create a new authentication process, follow these steps:

1. Add an authenticator (see page 48). The process refers to these authenticators, so it is helpful to have them already set up.
2. Click the **Authenticators** tab.
3. In the **Authentication Procs** section on the left, click **new**.
4. Enter a name for the new process, then select the authenticators that the process calls.

   If this is meant to be a two-factor authentication process, deselect **Password Only Authentication**.
5. Click **Next**, then click **Update** at the bottom of the page. Echidna immediately uses the new configuration.
6. Define whether a user attribute is to be used to select the preferred authentication method by editing the **User Selected Preferred Mechanism Attribute** section. This works if the user store includes a user attribute that stores the preferred authentication method.

   The attribute must contain the name of the user's preferred authentication method. To find this name, see Check the name of an authenticator on page 48.

   Update any of the following fields as well:

Parameter	Description
Attr Name	Name of the authentication-related user store attribute.
Label	Label to use for this attribute in the User Support console.
Description	Description of this attribute to show in the User Support console.
Pattern	A regex pattern that the attribute value must match to qualify for use with this mechanism. If this is blank, any non-empty value is acceptable.
Access	The method to be used to get this attribute updated. Possible values: • **USER_EDITABLE**: The end user is permitted to update this value from the Self Service console. • **USER_TRIGGERED_PROC**: The end user cannot directly edit the attribute, but can perform some action in the Self Service console that causes the attribute to be assigned an appropriate value. For example, registering for mCodeXpress or activating a YubiKey token. • **EXTERNAL_PROCESS**: This attribute is not updated at all via the Echidna support pages (either user or operator), but must be updated in the user store through some other process. • **SUPPORT_EDITABLE**: This value can be updated in the User Support console. • **SUPPORT_ASSIGNABLE**: In the User Support console, this value cannot be updated to an arbitrary value, but can trigger the attribute to be updated by assigning a registered token.

7. Define the **user roles**:

Parameter	Description
New External Name	This is the name of the role to be used in AVAILABLE condition tests or allowed-role definitions.
New Authentication Attributes Combination	For this role to be considered available to the user, the following conditions must be met: • **ONE_OF**: At least one of the listed authentication attributes must be present in the user record with a suitable value. • **ALL_OF**: All of the listed authentication attributes must be present in the user record with a suitable value
New Granting Groups Combination	For this role to be considered available to the user, the following conditions must be met: • **ONE_OF**: The user be a member of *at least one* of the listed groups. • **ALL_OF**: The user must be a member of *all* of the listed groups.

8. Define the **lockouts**:

Parameter	Description
Unlock Time (Minutes)	The time in minutes until a failed attempt is forgotten, allowing a new login attempt to occur.
Attempt Limit	The number of failed attempts within the time window that cause the user to be locked (prevented from authenticating).
Lockout User Identification	The expression (usually including user context attributes) that is used to index the stored failed login attempts.
Cache Name	If the lock-out state is to be stored in a shared cache, this is the name of that cache. If this is left blank, the state is stored in a local cache used only by this lockout-manager and server.
Result Type and Result Name	This determines which results are counted as *failed attempts* when processing user lockouts.

9. Define the process conditions. For information about these conditions, see How processes work on page 135.
10. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
11. Consider backing up the new configuration as a file. See Backup and recovery on page 130.

5.9 Salt mCodeXpress OTP

The mCodeXpress authenticator validates OTPs generated from the Salt mCodeXpress mobile soft token, supplied by Salt Group. The OTP can be validated in combination with the user-store password.

The following table describes the attributes that administrators can change:

Salt mCodeXpress Attribute	**Description**
Registration User Notice	The text that appears when a user registers their Salt mCodeXpress app with Echidna.
Token Link Attribute	The user attribute used to identify the registered Salt mCodeXPress tokens. This can be a single attribute such as **carLicense**, or an expression referencing one or more attributes (such as **${storeName}/${uid}**). Whichever attributes are referenced here, the same attributes should be identified in the **Auth Attributes** section above it, to enable appropriate management from the User Support console.
Passcode Length	The expected OTP length. By default, this is 8.
Passcode Ignore Characters	The characters to ignore in the passcodes that users enter. Anything matching this regular expression is removed from the OTP before validation. Default: space ( ) and dash (-).
Verification Counter Window	The number of the future sequence of OTPs to check against the user-input when validating the OTP.
Resynchronization Counter Window	The number of the future sequence of OTPs to check against the user-input when resynchronising the token.
Time Half-Window	The clock-drift to allow for when validating the time-based OTPs. The time is measured in clock ticks, where one tick equals 30 seconds. The default value is 9 ticks, which equals 270 seconds. This default value allows the Echidna server time to be up to 4 minutes and 30 seconds earlier or later than the predicted handset time. The difference between server time and handset time is predicted using previously successful OTP values.
Suspend Threshold	If greater than zero, the STATUSBIT_FAILTHRESHOLD bit (bit-1) is set on the token status when the fail-count exceeds this threshold. This is independent of any timed-lockout mechanism. The timed-lockout mechanisms automatically clear after the timeout, but the status bit is cleared only when an administrator enables the token again in the User Support console. If both are enabled, the suspend threshold should be set higher than the timed-lockout threshold.

Salt mCodeXpress Attribute	Description
Generated ID Length	The length of any new identifiers. Where the user attribute used to identify the Salt mCodeXpress tokens is not already populated, new identifiers can be generated. The new identifiers comply with any attribute pattern restrictions defined in the **Auth Attributes** section. This lets administrators include a token type in the identifier, which is useful for situations in which the same **tokenId** attribute is being shared to link to multiple alternative token types. For example, if the tokenId pattern is **MCX.{9}** and the generated ID length is **12**, the generated token identifiers look similar to **MCX329847381**.
Password Combination Method	The way that the user-store password and the Salt mCodeXpress OTP are to be combined in the passwords submitted by users. See Combining Echidna authentication methods with user store passwords on page 47 for more information.
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE**: With this option, a login challenge screen appears only if the submitted username and store password were correct. This may allow an attacker to try to guess user store passwords independently of the second factor (the Salt mCodeXpress OTP values) • **PASSCODE_THEN_PASSWORD**: The password is validated only after the Salt mCodeXpress OTP is validated. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected from password guessing and denial of service. • **PASSWORD_OPAQUE**: This option is not relevant for Salt mCodeXpress.
Registration Code Length	The length of the token registration code to be expected
Registration Code Ignore Characters	The characters to ignore in the registration codes that users enter. This is a regular expression.
Registration Code Normalization	The type of normalisation, if any, to be done to the registration code
Persistence	See How persistence works on page 142.
Phone Download Links	The download links of Salt mCodeXpress for each supported phone type. Salt mCodeXpress is available in various app stores.

Salt mCodeXpress Attribute	Description
Minimum Allowed Application ID and Maximum Allowed Application ID	The range of application IDs that can register for Salt mCodeXpress. Salt mCodeXpress registrations are accepted only if the application ID extracted from the **reg-key-check** value is between these values. Any tokens in this range are assumed to have the default behaviour, in which both time and event counters are used when generating OTPs or signature codes. Use Known Application IDs to list any exceptions that should still be allowed to register for Salt mCodeXpress.
Known Application IDs	Application IDs that are permitted to register for Salt mCodeXpress, but do not meet the criteria in Minimum Allowed Application ID and Maximum Allowed Application ID. The possible reasons for an application ID not fitting these criteria are: • An application ID is outside the normal range. • An application ID within that range but with non-default behaviour. These IDS are either event-only (no time base) or they are explicitly blocked from being registered.

5.10 SMS-delivered OTP

The SMS authenticator uses an SMS gateway to send messages. To use this authenticator, also configure Echidna to use a messaging (SMS) gateway, as described in Connect to a messaging gateway on page 75.

The SMS authenticator does the following steps:

1. Validates the submitted user ID against the resolved user store, and optionally validates the password (depending on the configured password validation order).
2. Generates a one-time passcode (OTP) and caches it in memory.
3. Sends the OTP through the SMS gateway to the mobile phone number from the user's store entry, and then challenges the user for the code.
4. Returns a challenge to the caller to enter the OTP from the phone.
5. On the follow-up call, validates the code that the user submitted against the generated OTP, and optionally validates the password against the user store if that wasn't done at step 1.

The trial Echidna license does not include the use of an SMS gateway. When the administrator upgrades Echidna from its evaluation license, they should request an SMS gateway license at the same time. For instructions, see Licensing for Echidna on page 132.

The following table describes the attributes that administrators can change:

SMS Attribute	Description
Registration User Notice	The text that appears when a user registers their SMS device with Echidna.
Token Link Attr	The attribute expression (or single attribute name) used to determine the token identifier from the user context.
Token Link Type	The type of linking between the user record and token record.
Token Link Multiplicity	Describe the possible multiplicity of user-to-token linking.
Generated ID Length	The length of identifiers generated during token registration if the tokenLinkAttr refers to user context attributes that are not yet populated. 0 indicates no identifiers to be generated.
Passcode Length	The length of the OTPs that this authenticator generates.
Passcode Ignore Characters Regex	Used to normalize the passcode before processing. Anything matching the regex is removed.
Password Combo	Allow validation of both the user-store password and the scheme-specific one-time-password.

SMS Attribute	Description
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE**: (Default) The password is validated before an SMS is generated, which can cut down on the number of generated SMS messages, but allow an attacker to try to guess user store passwords independently of the SMS codes. With this option, a login challenge screen appears only if the submitted username and store password were correct. • **PASSCODE_THEN_PASSWORD**: The password is validated only after the SMS OTP is validated. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected, but an attacker may cause more spurious SMS messages to be generated. • **PASSWORD_OPAQUE**: The password IS validated before an SMS is generated. If the password is incorrect no SMS is sent out, but the caller is still challenged for an OTP code. This protects the user store password better than PASSWORD_THEN_PASSCODE, and prevents spurious SMS messages better than PASSCODE_THEN_PASSWORD, but is potentially more confusing for legitimate users.
Challenge Type	Specify whether the token requires a challenge before the one-time-password can be generated, and if so how the challenge is generated.
Challenge Length	If the challenge is created randomly, how many digits are generated in each challenge
Suspend Threshold	If greater than zero, the 'STATUSBIT_FAILTHRESHOLD' bit will be set on the token status when the fail-count exceeds this threshold. This is independent of any timed-lockout mechanism.
Rechal On Empty OTP	When the submitted OTP is empty: true means an Access-Challenge will be sent to re-enter the OTP; false means an Access-Reject is immediately sent.
Token Record Persistence	This section controls how the SMS registration records are stored in the database and queried or updated.
Passcode Length	The number of characters in the generated OTPs.
Passcode Alphabet	The alphabet of characters to use in generating the OTPs.
Expiry Window	The number of seconds that a generated OTP remains valid. This also defines the time that an account remains locked due to incorrect password attempts.
Resend Window	The number of seconds during which a duplicate challenge call does not trigger a new OTP to be generated. This is usually set to a shorter time than the expiry window.

SMS Attribute	Description
Passcode-Verify Lock Threshold	The number of failed attempts to validate an OTP, within a single expiry window. When this number is exceeded, SMS authentication to that mobile number is locked until the expiry of that OTP.
SMS Text Message Format	The format of the SMS text message that contains the OTP. The default message is: `Your passcode is <passcode>` Use this field to change the message for just this authenticator. To change the message for all SMS authenticators, see **SMS_TEXT_FMT** in Change the format of SMS messages on page 80.
New Challenge Format Override	he format of the challenge message that the NAS (remote access server) presents to the user. The default message is: `enter the passcode sent to your registered mobile +<phonenumber>` Use this field to change the message for just this authenticator. To change the message for all SMS authenticators, see **SMS_CHAL_MSG** in Change the format of SMS messages on page 80.
Repeat Challenge Format Override	This is the challenge message displayed back to the login form. Overrides the RADIUS access-challenge reply-message format used when a previously sent SMS is still valid. The default message is: `enter the passcode sent to your registered mobile +<phonenumber>` If it is needed, this message can be changed. A more specific message could be provided, such as **The passcode previously sent to your registered mobile phone number is still active. Please enter this passcode.** Use this field to change the message for just this authenticator. To change the message for all SMS authenticators, see **SMS_CHAL_MSG** in Change the format of SMS messages on page 80.
	• • •

SMS Attribute	Description
Ignore Store Password	• **Unchecked**: (Default) If **Password Combo** is not set to PASSCODE_ONLY, the authenticator will validate the submitted store password, as described above. • **Checked**: (Not recommended) The password provided in that call can be ignored, and only the SMS OTP is checked. The SMS OTP is generated and sent to the user without validation of the store password. A dummy value may be sent as the password. This is useful in the unusual situation in which a system needs an SMS OTP without a user store password. The typical RADIUS interaction requires a password value to be provided in the initial call.
Auto Register	When checked, an end user can be automatically registered for SMS the first time they try to log in with that method provided the user store entry already has a mobile number, any other group and attribute restrictions are met, and the SMS license registration limits have not been exceeded.
Sender Proc Name	The name of the SMS sender process to use with this authenticator.

5.11 OATH HOTP (Generic)

The OATH HOTP authenticator validates OTPs generated using a token that implements the OATH SHA1-HOTP algorithm. The OTP can be validated in combination with the user-store password.

The following table describes the attributes that administrators can change:

OATH HOTP Attribute	Description
Token Link Attr	The user attribute used to identify the registered tokens. This can be a single attribute such as **carLicense**, or an expression referencing one or more attributes (such as **${storeName}/${uid}**). Whichever attributes are referenced here, the same attributes should be identified in the **Auth Attributes** section above it, to enable appropriate management from the User Support console.
Token Link Type	The type of linking between the user record and the token record: • **USER_ID:** The referenced attributes are part of the user identity and is always present. • **ASSIGNABLE:** The referenced attribute is dedicated to storing the serial number of the token. • **GENERATED:** The referenced attribute is populated with a randomly generated identifier when the user is linked to a token that doesn't have an intrinsic identifier or serial number.
Token Link Multiplicity	The mapping between users and tokens. This value is interpreted by the User Support console to enforce whether the one user can have more than one token assigned, and whether the same token can be assigned to more than one user. • ONE_TO_ONE (Recommended) • MANY_TO_ONE • ONE_TO_MANY • MANY_TO_MANY
Generated ID Length	The number of characters in the new identifier. This is used where the user attribute used to identify the OATH tokens is not already populated. If the value is 0, no ID is generated.
Pass Code Length	The expected OTP length. If using PASSCODE_WITH_PASSWORD for the password combination method, the first passcode-length characters are the passcode. If using PASSWORD_WITH_PASSCODE then the last passcode-length characters are the passcode. Limits: 6, 7, or 8 characters

OATH HOTP Attribute	Description
Passcode Ignore Characters Regex	The characters to ignore in the passcodes that users enter. Anything matching this regular expression is removed from the OTP. Default: space ( ) and dash (-).
Password Combo	The way that the user-store password and the OATH OTP are to be combined in the passwords submitted by users. This option can validate both the user-store password and the scheme-specific OTP. See Combining Echidna authentication methods with user store passwords on page 47 for more information.
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE:** With this option, a login challenge screen appears only if the submitted username and store password were correct. This may allow an attacker to try to guess user store passwords independently of the second factor (the OATH OTP values). • **PASSCODE_THEN_PASSWORD:** The password is validated only after the OATH OTP is validated. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected from password guessing and denial of service. • **PASSWORD_OPAQUE:** This option is not relevant for OATH.
Challenge Type	Specify whether the token requires a challenge before the one-time passcode can be generated, and if so how the challenge is generated. • NO_CHALLENGE • RANDOM_CHALLENGE • SERVER_OBTAINED_CHALLENGE • SERVER_TRIGGERED_CHALLENGE
Challenge Length	The number of digits that are generated in each challenge. This takes effect only if RANDOM_CHALLENGE is selected in Challenge Type.
Suspend Threshold	If greater than zero, the STATUSBIT_FAILTHRESHOLD bit (bit-1) is set on the token status when the fail-count exceeds this threshold. This is independent of any timed-lockout mechanism. The timed-lockout mechanisms automatically clear after the timeout, but the status bit is cleared only when an administrator enables the token again in the User Support console. If both are enabled, the suspend threshold should be set higher than the timed-lockout threshold.
Rechal On Empty OTP	Defines the behaviour if the submitted OTP is empty. • **Checked**: An access-challenge message is sent to re-enter the OTP • **Not checked**: An access-reject message is immediately sent

OATH HOTP Attribute	Description
Check Persistence	Immediately queries the selected data store to check for an existing table used by the OATH HOTP authenticator.
Create Persistence Table	Immediately creates a table in the token database, using the information further down the screen. This button immediately creates any tables that have been defined here but do not already exist.
Token Record Persistence	See How persistence works on page 142.
Mech Prov	The mechanism provider class. For HOTP, this is set to "au.com.saltgroup.token.hotp.HotpProvider".
Mech Config	The mechanism provider configuration to tune the behaviour of the HOTP validation. The default HOTP value is {"keyFactory":{"useSecretKeySpec":true,"keySize":160,"algorithm":"HmacSHA1"},"mac":{"transform":"HmacSHA1","init":null},"codeLength":6,"verifyWindow":10,"resynchWindow":500}
Allowed Unlock Algorithm	(Optional) Comma-separated list of algorithms that can be used to unlock the token. Any algorithm that is not listed here cannot be used to create a PIN unlock key (PUK). The first algorithm in the list is the default unlock algorithm. If this list is empty, any algorithm not on the **Denied Unlock Algorithm** list is permitted. For more information, see Allow OATH hardware tokens to be unlocked on page 85.
Denied Unlock Algorithm	(Optional) Comma-separated list of unlock algorithms that are not permitted to be used. If this token type contains more than one signature algorithm, specify them all here to prevent problems with non-repudiation. For more information, see Preventing problems with non-repudiation when generating PIN unlock keys (PUKs) on page 85.

5.11.1 Google Authenticator (HOTP)

The Google authenticator (HOTP) validates OTPs generated using the Google Authenticator app. This is very similar to OATH HOTP (Generic) authenticator, except for the token registration process. Google token registration is done through the Self Service console.

To create a Google Authenticator (HOTP) configuration, select the OATH HOTP (Generic) token type and modify the attributes accordingly. To enable registration of the Google Authenticator token in Self Service console, specify **USER_TRIGGERED_PROC** for each entry in **Auth Attributes**.

The standard Google Authenticator mobile application only generates OTPs of length 6, so the **Pass Code Length** field and the 'codeLength' attribute in the **Mech Config** field should be set to 6.

5.12 OATH TOTP

The OATH TOTP authenticator validates OTPs generated using a token that implements the OATH TOTP algorithm. The time-based OTP can be validated in combination with the user-store password.

The following table describes the attributes that administrators can change:

OATH TOTP Attribute	Description
Token Link Attr	The user attribute used to identify the registered tokens. This can be a single attribute such as **carLicense**, or an expression referencing one or more attributes (such as **${storeName}/${uid}**). Whichever attributes are referenced here, the same attributes should be identified in the **Auth Attributes** section above it, to enable appropriate management from the User Support console.
Token Link Type	The type of linking between the user record and the token record: • **USER_ID:** The referenced attributes are part of the user identity and is always present. • **ASSIGNABLE:** The referenced attribute is dedicated to storing the serial number of the token. • **GENERATED:** The referenced attribute is populated with a randomly generated identifier when the user is linked to a token that doesn't have an intrinsic identifier or serial number.
Token Link Multiplicity	The mapping between users and tokens. This value is interpreted by the User Support console to enforce whether the one user can have more than one token assigned, and whether the same token can be assigned to more than one user. • ONE_TO_ONE (Recommended) • MANY_TO_ONE • ONE_TO_MANY • MANY_TO_MANY
Generated ID Length	The number of characters in the new identifier. This is used where the user attribute used to identify the OATH tokens is not already populated. If the value is 0, no ID is generated.
Pass Code Length	The expected OTP length. If using PASSCODE_WITH_PASSWORD for the password combination method, the first passcode-length characters are the passcode. If using PASSWORD_WITH_PASSCODE then the last passcode-length characters are the passcode. Limits: 6, 7, or 8 characters
Passcode Ignore Characters Regex	The characters to ignore in the passcodes that users enter. Anything matching this regular expression is removed from the OTP. Default: space ( ) and dash (-).

OATH TOTP Attribute	Description
Password Combo	The way that the user-store password and the OATH OTP are to be combined in the passwords submitted by users. This option can validate both the user-store password and the scheme-specific OTP. See Combining Echidna authentication methods with user store passwords on page 47 for more information.
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE**: With this option, a login challenge screen appears only if the submitted username and store password were correct. This may allow an attacker to try to guess user store passwords independently of the second factor (the OATH OTP values). • **PASSCODE_THEN_PASSWORD**: The password is validated only after the OATH OTP is validated. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected from password guessing and denial of service. • **PASSWORD_OPAQUE**: This option is not relevant for OATH.
Challenge Type	Specify whether the token requires a challenge before the one-time passcode can be generated, and if so how the challenge is generated. • NO_CHALLENGE • RANDOM_CHALLENGE • SERVER_OBTAINED_CHALLENGE • SERVER_TRIGGERED_CHALLENGE
Challenge Length	The number of digits that are generated in each challenge. This takes effect only if RANDOM_CHALLENGE is selected in **Challenge Type**.
Suspend Threshold	If greater than zero, the STATUSBIT_FAILTHRESHOLD bit (bit-1) is set on the token status when the fail-count exceeds this threshold. This is independent of any timed-lockout mechanism. The timed-lockout mechanisms automatically clear after the timeout, but the status bit is cleared only when an administrator enables the token again in the User Support console. If both are enabled, the suspend threshold should be set higher than the timed-lockout threshold.
Rechal On Empty OTP	Defines the behaviour if the submitted OTP is empty. • **Checked**: An access-challenge message is sent to re-enter the OTP • **Not checked**: An access-reject message is immediately sent
Check Persistence	Immediately queries the selected data store to check for an existing table used by the OATH TOTP authenticator.

OATH TOTP Attribute	Description
Create Persistence Table	Immediately creates a table in the token database, using the information further down the screen. This button immediately creates any tables that have been defined here but do not already exist.
Token Record Persistence	See How persistence works on page 142.
Mech Prov	The mechanism provider class. For TOTP, this is set to "au.com.saltgroup.token.totp.TotpProvider".
Mech Config	The mechanism provider configuration to tune the behaviour of the TOTP validation. The default TOTP value is {"keyFactory":{"useSecretKeySpec":true,"keySize":160,"algorithm":"HmacSHA1"},"mac":{"transform":"HmacSHA1","init":null},"codeLength":8,"timeStep":30,"zeroTime":0,"verifyWindow":{"notBefore":-5,"notAfter":5},"firstUseWindow":{"notBefore":-20,"notAfter":20},"resynchWindow":{"notBefore":-100,"notAfter":100}}
Allowed Unlock Algorithm	(Optional) Comma-separated list of algorithms that can be used to unlock the token. Any algorithm that is not listed here cannot be used to create a PIN unlock key (PUK). The first algorithm in the list is the default unlock algorithm. If this list is empty, any algorithm not on the **Denied Unlock Algorithm** list is permitted. For more information, see Allow OATH hardware tokens to be unlocked on page 85.
Denied Unlock Algorithm	(Optional) Comma-separated list of unlock algorithms that are not permitted to be used. If this token type contains more than one signature algorithm, specify them all here to prevent problems with non-repudiation. For more information, see Preventing problems with non-repudiation when generating PIN unlock keys (PUKs) on page 85.

5.12.1 Google Authenticator (TOTP)

The Google authenticator (TOTP) validates OTPs generated using the Google Authenticator app. This is very similar to OATH TOTP (Generic) authenticator, except for the token registration process. Google token registration is done through the Self Service console.

To create a Google Authenticator (TOTP) configuration, select the OATH TOTP token type and modify the attributes accordingly. To enable registration of the Google Authenticator token in Self Service console, specify **USER_TRIGGERED_PROC** for each entry in **Auth Attributes**.

The standard Google Authenticator mobile application only generates OTPs of length 6, so the **Pass Code Length** field and the 'codeLength' attribute in the **Mech Config** field should be set to 6.

5.13 OATH OCRA

The OATH OCRA authenticator validates OTPs and signatures generated using a token that implements the OATH OCRA algorithms. The time or event based OTPs can be validated in combination with the user-store password. The signature codes are only validated via web service calls (not as part of regular authentication).

The following table describes the attributes that administrators can change:

OATH OCRA Attribute	Description
Token Link Attr	The user attribute used to identify the registered tokens. This can be a single attribute such as **carLicense**, or an expression referencing one or more attributes (such as **${storeName}/${uid}**). Whichever attributes are referenced here, the same attributes should be identified in the **Auth Attributes** section above it, to enable appropriate management from the User Support console.
Token Link Type	The type of linking between the user record and the token record: • **USER_ID:** The referenced attributes are part of the user identity and is always present. • **ASSIGNABLE:** The referenced attribute is dedicated to storing the serial number of the token. • **GENERATED:** The referenced attribute is populated with a randomly generated identifier when the user is linked to a token that doesn't have an intrinsic identifier or serial number.
Token Link Multiplicity	The mapping between users and tokens. This value is interpreted by the User Support console to enforce whether the one user can have more than one token assigned, and whether the same token can be assigned to more than one user. • ONE_TO_ONE (Recommended) • MANY_TO_ONE • ONE_TO_MANY • MANY_TO_MANY
Generated ID Length	The number of characters in the new identifier. This is used where the user attribute used to identify the OATH tokens is not already populated. If the value is 0, no ID is generated.
Pass Code Length	The expected OTP length. If using PASSCODE_WITH_PASSWORD for the password combination method, the first passcode-length characters are the passcode. If using PASSWORD_WITH_PASSCODE then the last passcode-length characters are the passcode. Limits: 6-10 characters
Passcode Ignore Characters Regex	The characters to ignore in the passcodes that users enter. Anything matching this regular expression is removed from the OTP. Default: space ( ) and dash (-).

OATH OCRA Attribute	Description
Password Combo	The way that the user-store password and the OATH OTP are to be combined in the passwords submitted by users. This option can validate both the user-store password and the scheme-specific OTP. See Combining Echidna authentication methods with user store passwords on page 47 for more information.
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE**: With this option, a login challenge screen appears only if the submitted username and store password were correct. This may allow an attacker to try to guess user store passwords independently of the second factor (the OATH OTP values). • **PASSCODE_THEN_PASSWORD**: The password is validated only after the OATH OTP is validated. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected from password guessing and denial of service. • **PASSWORD_OPAQUE**: This option is not relevant for OATH.
Challenge Type	Specify whether the token requires a challenge before the one-time passcode can be generated, and if so how the challenge is generated. • NO_CHALLENGE • RANDOM_CHALLENGE • SERVER_OBTAINED_CHALLENGE • SERVER_TRIGGERED_CHALLENGE
Challenge Length	The number of digits that are generated in each challenge. This takes effect only if RANDOM_CHALLENGE is selected in Challenge Type.
Suspend Threshold	If greater than zero, the STATUSBIT_FAILTHRESHOLD bit (bit-1) is set on the token status when the fail-count exceeds this threshold. This is independent of any timed-lockout mechanism. The timed-lockout mechanisms automatically clear after the timeout, but the status bit is cleared only when an administrator enables the token again in the User Support console. If both are enabled, the suspend threshold should be set higher than the timed-lockout threshold.
Rechal On Empty OTP	Defines the behaviour if the submitted OTP is empty. • **Checked**: An access-challenge message is sent to re-enter the OTP • **Not checked**: An access-reject message is immediately sent
Check Persistence	Immediately queries the selected data store to check for an existing table used by the OATH OCRA authenticator.

OATH OCRA Attribute	Description
Create Persistence Table	Immediately creates a table in the token database, using the information further down the screen. This button immediately creates any tables that have been defined here but do not already exist.
Token Record Persistence	See How persistence works on page 142.
Allowed Unlock Algorithm	(Optional) Comma-separated list of algorithms that can be used to unlock the token. Any algorithm that is not listed here cannot be used to create a PIN unlock key (PUK). The first algorithm in the list is the default unlock algorithm. If this list is empty, any algorithm not on the **Denied Unlock Algorithm** list is permitted. For more information, see Allow OATH hardware tokens to be unlocked on page 85.
Denied Unlock Algorithm	(Optional) Comma-separated list of unlock algorithms that are not permitted to be used. If this token type contains more than one signature algorithm, specify them all here to prevent problems with non-repudiation. For more information, see Preventing problems with non-repudiation when generating PIN unlock keys (PUKs) on page 85.

5.14 OATH HOTP (YubiKey)

The OATH HOTP (YubiKey) authenticator validates OTPs generated using a YubiKey token. The OTP can be validated in combination with the user-store password.

This is very similar to the OATH HOTP (Generic) authenticator, except for the token registration process. For YubiKey tokens, no PSKC token import process is needed and the OATH HOTP seed material is generated and imported at the time each YubiKey token is activated.

The following table describes the attributes that administrators can change:

OATH HOTP (YubiKey) Attribute	**Description**
Token Link Attr	The user attribute used to identify the registered tokens. This can be a single attribute such as **carLicense**, or an expression referencing one or more attributes (such as **${storeName}/${uid}**). Whichever attributes are referenced here, the same attributes should be identified in the **Auth Attributes** section above it, to enable appropriate management from the User Support console.
Token Link Type	The type of linking between the user record and the token record: • **USER_ID:** The referenced attributes are part of the user identity and is always present. • **ASSIGNABLE:** The referenced attribute is dedicated to storing the serial number of the token. • **GENERATED:** The referenced attribute is populated with a randomly generated identifier when the user is linked to a token that doesn't have an intrinsic identifier or serial number.
Token Link Multiplicity	The mapping between users and tokens. This value is interpreted by the User Support console to enforce whether the one user can have more than one token assigned, and whether the same token can be assigned to more than one user. • ONE_TO_ONE (Recommended) • MANY_TO_ONE • ONE_TO_MANY • MANY_TO_MANY
Generated ID Length	The number of characters in the new identifier. This is used where the user attribute used to identify the OATH tokens is not already populated. If the value is 0, no ID is generated.

OATH HOTP (YubiKey) Attribute	Description
Pass Code Length	The expected OTP length. If using PASSCODE_WITH_PASSWORD for the password combination method, the first passcode-length characters are the passcode. If using PASSWORD_WITH_PASSCODE then the last passcode-length characters are the passcode. Limits: 6, 7, or 8 characters
Passcode Ignore Characters Regex	The characters to ignore in the passcodes that users enter. Anything matching this regular expression is removed from the OTP. Default: space ( ) and dash (-).
Password Combo	The way that the user-store password and the OATH OTP are to be combined in the passwords submitted by users. • PASSCODE_ONLY • PASSCODE_WITH_PASSWORD • PASSWORD_WITH_PASSCODE • PASSWORD_THEN_PASSCODE This option can validate both the user-store password and the scheme-specific OTP. See Combining Echidna authentication methods with user store passwords on page 47 for more information.
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE**: With this option, a login challenge screen appears only if the submitted username and store password were correct. This may allow an attacker to try to guess user store passwords independently of the second factor (the OTP values). • **PASSCODE_THEN_PASSWORD**: The password is validated only after the OTP is validated. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected from password guessing and denial of service. • **PASSWORD_OPAQUE**: This option is not relevant for YubiKey.
Challenge Type	Specify whether the token requires a challenge before the one-time passcode can be generated, and if so how the challenge is generated. • NO_CHALLENGE • RANDOM_CHALLENGE • SERVER_OBTAINED_CHALLENGE • SERVER_TRIGGERED_CHALLENGE

OATH HOTP (YubiKey) Attribute	Description
Challenge Length	The number of digits that are generated in each challenge. This takes effect only if RANDOM_CHALLENGE is selected in Challenge Type.
Suspend Threshold	If greater than zero, the STATUSBIT_FAILTHRESHOLD bit (bit-1) is set on the token status when the fail-count exceeds this threshold. This is independent of any timed-lockout mechanism. The timed-lockout mechanisms automatically clear after the timeout, but the status bit is cleared only when an administrator enables the token again in the User Support console. If both are enabled, the suspend threshold should be set higher than the timed-lockout threshold.
Rechal On Empty OTP	Defines the behaviour if the submitted OTP is empty. • **Checked**: An access-challenge message is sent to re-enter the OTP • **Not checked**: An access-reject message is immediately sent
Check Persistence	Immediately queries the selected data store to check for an existing table used by the OATH HOTP (Yubikey) authenticator.
Create Persistence Table	Immediately creates a table in the token database, using the information further down the screen. This button immediately creates any tables that have been defined here but do not already exist.
Token Record Persistence	See How persistence works on page 142.
Verify Window	The number of the future sequence of OTPs to check against the user-input when validating the OTP.
First Use Window	The number of forward event counter values used to check when validating an OTP value and it is the first time the token has been used.
Resynch Window	The number of the future sequence of OTPs to check against the user-input when resynchronising the token.

5.15 Limited-use temporary password

The temporary password authenticator is a form of static password authentication. It is designed for the case where the user has a standard authentication mechanism (token or SMS) that is temporarily unavailable and the user has an urgent need to still access the system. In this case, support staff can use the User Support console to generate a new random temporary password for that user and communicate it to the user.

The password can only be used a fixed number of times (default once). Whether is was used or not, the password expires after a fixed time period (default 1 day). After the password is used or expired, the user reverts to their original authentication mechanism.

The following table describes the attributes that administrators can change:

Temporary Password Attribute	**Description**
Token Link Attr	The user attribute used to identify the registered tokens. This can be a single attribute such as **carLicense**, or an expression referencing one or more attributes (such as **${storeName}/${uid}**). Whichever attributes are referenced here, the same attributes should be identified in the **Auth Attributes** section above it, to enable appropriate management from the User Support console.
Token Link Type	The type of linking between the user record and the token record: • **USER_ID:** The referenced attributes are part of the user identity and is always present. • **ASSIGNABLE:** The referenced attribute is dedicated to storing the serial number of the token. • **GENERATED:** The referenced attribute is populated with a randomly generated identifier when the user is linked to a token that doesn't have an intrinsic identifier or serial number.
Token Link Multiplicity	The mapping between users and tokens. This value is interpreted by the User Support console to enforce whether the one user can have more than one token assigned, and whether the same token can be assigned to more than one user. • ONE_TO_ONE (Recommended) • MANY_TO_ONE • ONE_TO_MANY • MANY_TO_MANY
Generated ID Length	The number of characters in the new identifier. This is used where the user attribute used to identify the OATH tokens is not already populated. If the value is 0, no ID is generated.

Temporary Password Attribute	Description
Pass Code Length	The expected OTP length. If using PASSCODE_WITH_PASSWORD for the password combination method, the first passcode-length characters are the passcode. If using PASSWORD_WITH_PASSCODE then the last passcode-length characters are the passcode. Limits: 6, 7, or 8 characters
Passcode Ignore Characters Regex	The characters to ignore in the passcodes that users enter. Anything matching this regular expression is removed from the OTP. Default: space ( ) and dash (-).
Password Combo	The way that the user-store password and the OATH OTP are to be combined in the passwords submitted by users. • PASSCODE_ONLY • PASSCODE_WITH_PASSWORD • PASSWORD_WITH_PASSCODE • PASSWORD_THEN_PASSCODE This option can validate both the user-store password and the scheme-specific OTP. See Combining Echidna authentication methods with user store passwords on page 47 for more information.
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE**: With this option, a login challenge screen appears only if the submitted username and store password were correct. This may allow an attacker to try to guess user store passwords independently of the second factor (the temporary password). • **PASSCODE_THEN_PASSWORD**: The password is validated only after the temporary password is validated. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected from password guessing and denial of service. • **PASSWORD_OPAQUE**: This option is not relevant for the Temporary Password authenticator.
Challenge Type	Specify whether the token requires a challenge before the one-time passcode can be generated, and if so how the challenge is generated. • NO_CHALLENGE • RANDOM_CHALLENGE • SERVER_OBTAINED_CHALLENGE • SERVER_TRIGGERED_CHALLENGE

Temporary Password Attribute	Description
Challenge Length	The number of digits that are generated in each challenge. This takes effect only if RANDOM_CHALLENGE is selected in Challenge Type.
Suspend Threshold	If greater than zero, the STATUSBIT_FAILTHRESHOLD bit (bit-1) is set on the token status when the fail-count exceeds this threshold. This is independent of any timed-lockout mechanism. The timed-lockout mechanisms automatically clear after the timeout, but the status bit is cleared only when an administrator enables the token again in the User Support console. If both are enabled, the suspend threshold should be set higher than the timed-lockout threshold.
Rechal On Empty OTP	Defines the behaviour if the submitted OTP is empty. • **Checked**: An access-challenge message is sent to re-enter the OTP • **Not checked**: An access-reject message is immediately sent
Check Persistence	Immediately queries the selected data store to check for an existing table used by the temporary password authenticator.
Create Persistence Table	Immediately creates a table in the token database, using the information further down the screen. This button immediately creates any tables that have been defined here but do not already exist.
Token Record Persistence	See How persistence works on page 142.
Max Validity Time	The time, in seconds, that the temporary password is valid for (default 86400 for 24 hours).
Max Failure Count	The number of incorrect password attempts before the temporary password is suspended.
Max Use Count	The number of times the password can be used for a successful login (default 1 for single-use).

5.16 Password only

The password-only authenticator validates only the user's password from the user-store. It does not provide second-factor authentication. For example, use this method in these situations:

- When testing RADIUS timeout settings or load-balancing.
- To protect less-sensitive material where two-factor authentication is not needed, as when a user is already in the LAN.

The following table describes the attributes that administrators can change:

Password-only Attribute	**Description**
Authentication delay	Introduce an artificial delay (number of seconds) before validating the store credential. For testing purposes only.

5.17 CAPTCHA images

Guards against automated attempts to break the password-only method. If the user has entered a wrong password too many times, send a CAPTCHA request.

The following table describes the attributes that administrators can change:

CAPTCHA Attribute	Description
Activation Threshold	The number of failed login attempts before CAPTCHA images are activated.
Passcode Length	The number of characters in the generated OTPs.
Passcode Alphabet	The characters that can be included in the OTPs. The default is digits only.
Expiry Window	The number of seconds that a generated OTP remains valid. This also defines the time that an account remains locked due to incorrect password attempts.
Challenge on wrong CAPTCHA OTP	Whether to send an ACCESS-CHALLENGE response when the CAPTCHA OTP is incorrect (to immediately challenge for re-entry of the OTP), or just to send an ACCESS-REJECT.
CAPTCHA OTP Verify Lock Threshold	The number of failed attempts to validate a specific OTP that will trigger a lockout of CAPTCHA authentication for that user until the expiry of that OTP.
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE**: With this option, a login challenge screen appears only if the submitted username and store password were correct. This may allow an attacker to try to guess user store passwords independently of the second factor (the CAPTCHA code). • **PASSCODE_THEN_PASSWORD**: The password is validated only after the CAPTCHA code is validated. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected from password guessing and denial of service. • **PASSWORD_OPAQUE**: This option is not relevant for CAPTCHA.
Image Width	The width of the image displayed to the user.
Image Height	The height of the image displayed to the user.
Response Image Attribute	The response attribute to put the CAPTCHA image's bytes in.
CAPTCHA Image Generation Steps	The set of processing steps used to generate the CAPTCHA image.

5.18 Connect to a messaging gateway

Echidna can use messaging gateways such as Clickatell to send SMS messages to authenticating users. This is useful for Salt mCodeXpress (page 51) and SMS OTPs (page 54).

The trial Echidna license allows Echidna to use the file gateway for testing. The evaluation license does not include the use of a messaging gateway. When the administrator uses the instructions in Licensing for Echidna (see page 132), they should request a messaging gateway license at the same time.

5.18.1 Add a new messaging gateway

To add a messaging gateway to Echidna, follow these steps:

1. Log in to the Administration console.
2. Click the **SMS Gateways** tab.
3. In the **SMS Gateways** section on the left, click **new**.
4. Enter the name of the new messaging gateway and select the type of gateway:
   - File-storage gateway
   - HTTP-invoked gateway
   - SMTP gateway (email)
5. Click **Next**.
   The new gateway configuration is created with default values.
6. Change some or all of the default values. Use the descriptions in the following tables.
7. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
8. Consider backing up the new configuration as a file. See Backup and recovery on page 130.

5.18.2 File storage gateway

The following table lists the parameters for a file storage gateway:

File-storage gateway attribute	Description
Params	Not applicable to the file storage gateway.
Username *and* Password	Not applicable to the file storage gateway.
Message Folder	The folder on the Echidna server where the message files are created.
Message File	The naming pattern for the SMS message file. This may include variable references such as the following: • **${mobile}**: The message recipient phone number or email address • **${userContext.uid}**: The username of the target user. • **${TS_Date}**: The current time, represented as a timestamp which is formatted as **yyyyMMdd_HHmmss_SSS**. The default pattern is ${mobile}/${TS_Date}.sms., which gives filenames similar to the following: **61425555123/20141112_094501_201.sms**.
Message Content	The format for the content that is written to the message file, which usually includes a reference to ${messagetext} and possibly to ${mobile}.
Append	If the message file format includes ${TS_Date}, that is usually enough to ensure the file is unique for each message. If the target message file *does* already exist, the *append* setting controls how the message is added to the existing file: • **Selected**: (Default) Add the message to the end of the file, leaving the previous messages in the file unchanged. • **Not selected**: Overwrite the message file each time a new message is created. The most common strategy for the message files that are actually being processed further is to have one file per message, as defined here: • ${mobile}/${TS_Date}.sms) For testing, one file per user (${mobile}.sms) may be more convenient, and setting append=false allows only the latest message to be maintained.

5.18.3 HTTP-invoked SMS gateway

The following table lists the parameters for an HTTP-invoked SMS gateway:

HTTP-invoked SMS gateway attribute	**Description**
Params	The list of generic key-value configuration pairs. These support the plug-in of custom gateway types that are not a part of the core Echidna libraries.
Username *and* Password	The credentials used by Echidna to bind to the SMS gateway. The format of the username is determined by the remote gateway, but typically appears like a simple username or an email address.
Send Message Service Call	The HTTP service call configuration used to send SMS messages via the gateway. Configure the following: • Service URL • HTTP method (POST or GET) • HTTP authentication type (BASIC, DIGEST, or NONE) • The HTTP headers and request query parameters to attach to the call
Query Message Status Service Call	The HTTP service call configuration used to query the status of a previously sent SMS message. Configure the following: • Service URL • HTTP method (POST or GET) • HTTP authentication type (BASIC, DIGEST, or NONE) • The HTTP headers and request query parameters to attach to the call

5.18.4 SMTP SMS gateway

The following table lists the parameters for an SMTP SMS gateway:

SMTP SMS gateway attribute	**Description**
Params	Not applicable to the SMTP SMS gateway.
Username *and* Password	The credentials used by Echidna to bind to the SMS gateway. The format of the username is determined by the remote gateway, but typically appears like a simple username or an email address.
Sender Email Address *and* Sender Friendly Name	This controls the apparent sender's email address and display name, in email messages generated by Echidna.

Message Body	The format of the email messages' headers and content in accordance with RFC5321 (SMTP). At a minimum this must include a **To:** header and a reference to the SMS text (${smstext}) in the body or subject line. The default format is:
	`From: "${senderFriendlyName}" <${senderEmailAddress}>`
	`To: <${recipient}>`
	`Date: ${HTTP_Date}`
	`Subject: Your One-Time Password`
	
	`Your one-time password is`
	`${smstext}`
	Available reference variables are any user attributes (such as **${userContext.displayName}**) as well as the following:
	• **${ smsrecipient} or ${recipient**}: The email address of the intended message recipient
	• **${smstext}** or **${messagetext}**: The message text (including the OTP value) as generated by the SMS OTP authenticator (using the SMS_TEXT_FMT message format).
	• **${senderFriendlyName}**: The sender friendly name from the **Sender Friendly Name** field on the current page.
	• **${senderEmailAddress}**: The sender email address from the **Sender Email Address** field on the current page.
	• **${HTTP_Date}**: The current timestamp in HTTP date format (EEE, d MMM yyyy HH:mm:ss 'GMT').

5.18.5 Configure the SMS process

Echidna allows administrators to configure one or more SMS processes.

The SMS process tells Echidna how to interpret the stored mobile number of a user, and subsequently selects the SMS gateway to use to send an SMS to that user.

When adding a new SMS gateway to Echidna, make sure that the SMS process includes the new SMS gateway.

To configure an SMS process:

1. Log in to the Administration console.
2. Click the **SMS Gateways** tab.
3. Under **Processes** on the left, click **Send SMS Process**.
4. Add, remove or edit the conditions of the process.
5. Click on the **Update** button at the bottom.

5.18.6 Change the format of SMS messages

By default, SMS messages sent by Echidna use this format:

```
Your passcode is nnnn-nnnn
```

This default message can be overridden by editing the SMS Text Message Format in each instance of the SMS-delivered OTP authenticator. For a description, see SMS-delivered OTP on page 54.

Administrators can also change the default message format. This affects all existing SMS authenticators that do not override the default message format.

If Echidna includes two instances of the SMS authenticator, each can send its own message. This is useful when one SMS authenticator actually sends emails, and the other sends SMS messages.

To change the new-OTP message sent to users:

1. Log in to the Administration console.
2. Click the **Msg Specs** tab. This page lists the formats used in various messages sent to users and logs.
3. Scroll down to find the **SMS_TEXT_FMT** section.
4. Enter a new message. This message can contain the following parameters:
   - **{0}**: The entire OTP value
   - **{1}**: The first set of four digits in the OTP
   - **{2}**: The second set of four digits in the OTP
   - **{3}**: The third set of four digits in the OTP (where applicable)
   - **{4}**: The fourth set of four digits in the OTP (where applicable)

   By default, the message uses the following text, where nnnn-nnnn represents an eight-digit OTP:
   - **Actual message**: Your passcode is nnnn-nnnn
   - **Text in this box**: Your passcode is {1}-{2}
5. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
6. Consider backing up the new configuration as a file.

The following table lists the formats for SMS messages:

Name of format	Type of message	Parameters
SMS_CHAL_MSG	This format is used for the challenge message that the NAS (remote access server) presents to the user during login.	• **{0}**: The MOBILE_ATTVALUE mobile phone number or token identifier from the user-context • **{1}**: MASKED_MOBILE_ATTVALUE partially masked mobile phone number from the user-context

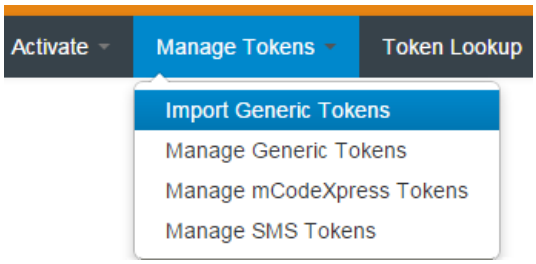
Name of format	Type of message	Parameters
SMS_ALREADY_MSG	Similar to SMS_CHAL_MSG, this format is used for the challenge message that the NAS (remote access server) presents to the user during login, when a previous SMS is still valid.	• **{0}**: The MOBILE_ATTVALUE mobile phone number or token identifier from the user-context • **{1}**: MASKED_MOBILE_ATTVALUE partially masked mobile phone number from the user-context
SMS_EXPIRED_MSG	This format is used for the reply-message returned by Echidna when validating a submitted OTP and the previously generated OTP has either expired or already been used once. The NAS (or other login page) often doesn't display this to the user on failed login, but may record it in its own logs.	• **{0}**: The MOBILE_ATTVALUE mobile phone number or token identifier from the user-context
SMS_LOCKED_MSG	This format is used for the reply-message returned by Echidna when attempting to generate or validate an OTP, but the SMS authentication has put the user into a timed lockout. This occurs when the guess-limit within the time-window for the given phone number has been exceeded, so the user has to wait for the time to expire before they can attempt authentication again.	• **{0}**: The MOBILE_ATTVALUE mobile phone number or token identifier from the user-context
SMS_NOWLOCKED_LOG	This format is used for the reply-message returned by Echidna when attempting to validate an OTP, but the OTP value is incorrect and the lockout threshold has just been hit. Further attempts to log in return the SMS_LOCKED_MSG message until the timeout expires.	• **{0}**: The MOBILE_ATTVALUE mobile phone number or token identifier from the user-context
SMS_TEXT_FMT	SMS to a user that contains the user's new OTP	• **{0}**: The entire OTP value • **{1}**: The first set of four digits in the OTP • **{2}**: The second set of four digits in the OTP • **{3}**: The third set of four digits in the OTP • **{4}**: The fourth set of four digits in the OTP.

Name of format	Type of message	Parameters
SMS_FAILURE_MSG	This format is used for the reply-message returned by Echidna when attempting to generate an OTP, but the SMS gateway was unable to accept the message delivery request.	• **{0}**: MOBILE_ATTVALUE mobile phone number or token identifier from the user-context • **{1}**: ERRORTEXT dependant-service error text • **{2}**: MASKED_MOBILE_ATTVALUE partially masked mobile phone number from the user-context • **{3}**: USERNAME the resolved full user name, or the username from the request

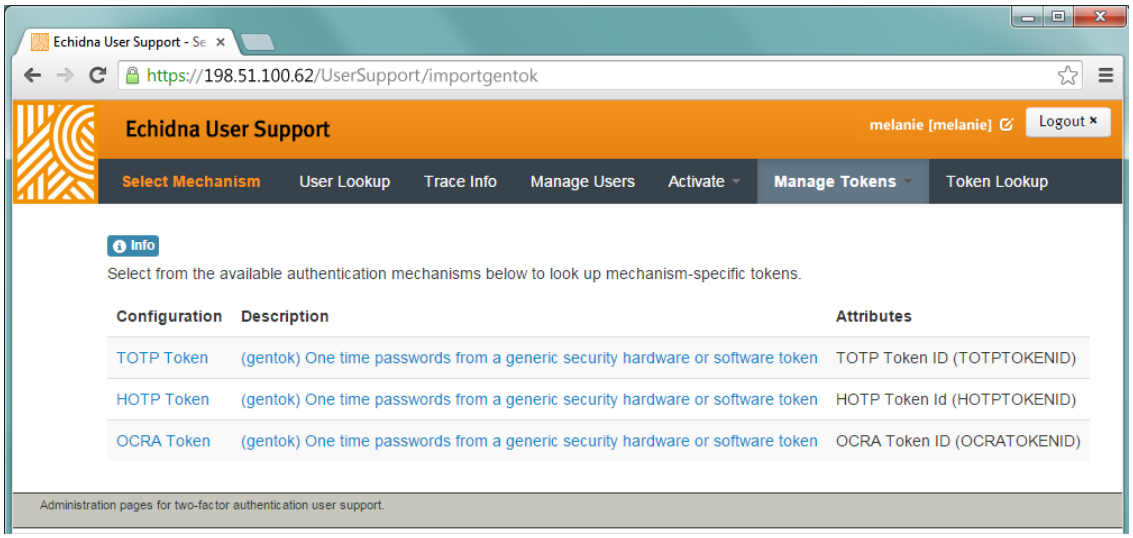
5.19 Import OATH token seed material

Echidna uses the seed files to create the records for each OTP for each OATH token. These records are used to calculate the correct OTPs.

1. Log in to the **User Support** console.
2. Click **Import Generic Tokens**:

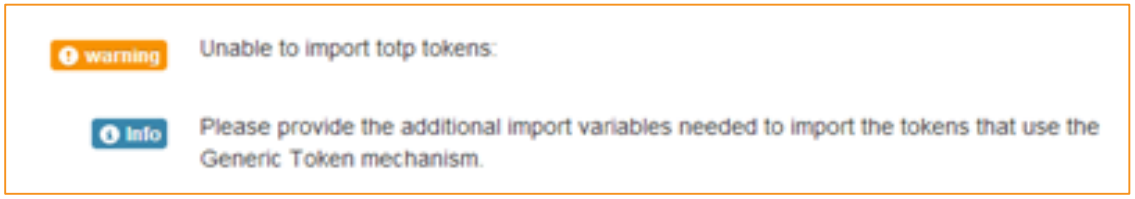


If more than one type of OATH token is available, they are all listed:



3. Click on the name of the token type.
4. Click **Choose File** and navigate to the seed file.
5. Select a **Key Translator** and a **Key Verifier**.

   The key translator is needed if the import file has encrypted seeds (recommended for production systems). The key verifier is optional but recommended.
6. Click **Load**. Echidna checks the format of the PSKC file.
7. If extra information is needed, Echidna asks for it. In the example below, a password is required:



8. For test tokens using a PBE-AES key, type the plaintext password in the field.

   For production tokens, paste the contents of the accompanying transport key .asc file, which looks similar to the following:

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v2.0.22 (MingW32)
hIwDdMxBms6QDTcBA/9+BlY+FNyD70LBegjOLUzjAd4labG5605w4teajHGXy2n7
g2YmWhBZEwTB6UgCxpbFyrpKnM5yEQWUkBt5nO75gQ==
=3DiF
-----END PGP MESSAGE-----
```

9. Echidna checks that the data is valid and at least one token can be imported.

If the import is successful, a summary of the result appears with links to manage the first few imported tokens.

5.20 Allow OATH hardware tokens to be unlocked

This section applies to OATH tokens that require a PIN.

If an OATH hardware token receives a series of invalid PIN entries, the token is locked. When the user enters the correct PIN unlock key (PUK), the token is unlocked and can be used again.

OATH tokens with PINS can support one of the following methods for obtaining the PUK:

- Some OATH tokens generate a challenge code, which the user enters into Echidna. Echidna uses this challenge code to generate the PUK.
- Other OATH tokens do not provide a challenge code. These tokens require a PUK generated by Echidna.

In both cases, Echidna must use the correct algorithm to generate the PUK.

5.20.1 Preventing problems with non-repudiation when generating PIN unlock keys (PUKs)

Each token has at least one algorithm that is used to create signatures. A token's signature algorithm must not be used to create a PUK. If it were, the PUK could be used to sign transactions.

To prevent this problem, Echidna automatically checks the seed file to identify the algorithm that is listed as the token's signature algorithm. Echidna then refuses to create a PUK for that token with that algorithm.

If a token has only one signature algorithm, this works well. In this case, the administrator does not need to specify which algorithms can and cannot be used to create PUKs.

If a token has more than one signature algorithm, this is not sufficient, because a seed file can list only one signature algorithm. In this case, use the **Allowed Unlock Algorithm** or **Denied Unlock Algorithm** boxes to specify algorithms that can and cannot be used.

These boxes are available in the configuration for each OATH authenticator. See the descriptions on pages 60, 63, and 66 for more information.

5.20.2 Create a web interface for generating PIN unlock keys (PUKs)

Echidna's web services API contains the **generateUnlockCode** service for each OATH authenticator type. Use this service to create a web interface that lets users generate PUKs.

Chapter 6: Brokering authentication

This chapter describes how to use Echidna to broker authentication requests to another authentication server.

6.1 Remote RADIUS

The remote RADIUS authenticator is similar in concept to the Salt mCodeXpress or OATH token authenticator, but it makes use of a remote RADIUS server to validate the OTPs. This allows the Echidna server to act as a RADIUS proxy server.

The following table describes the attributes that administrators can change:

RADIUS Attribute	Description
User ID Link Attribute	The user attribute(s) used to format the User-Name attribute in the remote RADIUS request.
Pass Code Length	The expected OTP length. If using PASSCODE_WITH_PASSWORD for the password combination method, the first passcode-length characters are the passcode. If using PASSWORD_WITH_PASSCODE then the last passcode-length characters are the passcode. Limits: 6, 7, or 8 characters
Passcode Ignore Characters	The characters to ignore in the passcodes that users enter. Anything matching this regular expression is removed from the OTP. Default: space ( ) and dash (-).
Generated ID length	The number of characters in the new identifier. This is used where the user attribute used to identify the OTPs is not already populated. If the value is 0, no ID is generated.

RADIUS Attribute	Description
Password Combination Method	The way that the user-store password and the OTP are to be combined in the passwords submitted by users. This option can validate both the user-store password and the scheme-specific OTP. See Combining Echidna authentication methods with user store passwords on page 47 for more information.
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE**: With this option, a login challenge screen appears only if the submitted username and store password were correct. This may allow an attacker to try to guess user store passwords independently of the second factor (the remote RADIUS server's OTP). • **PASSCODE_THEN_PASSWORD**: The store password is validated only after the OTP is validated via the remote RADIUS server. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected from password guessing and denial of service. • **PASSWORD_OPAQUE**: This option is not relevant for RADIUS.
Challenge Type	Specify whether the token requires a challenge before the one-time passcode can be generated, and if so how the challenge is generated. • NO_CHALLENGE • RANDOM_CHALLENGE • SERVER_OBTAINED_CHALLENGE • SERVER_TRIGGERED_CHALLENGE
Challenge Length	The number of digits that are generated in each challenge. This takes effect only if RANDOM_CHALLENGE is selected in Challenge Type.
Remote Request Retry Count	The number of times to attempt sending a remote RADIUS request after a timeout. When this number is exceeded, no more attempts are made for that request.
Remote Server Routing Algorithm	Which order to try invoking the available remote servers • **FAILOVER**: Sends requests to a single remote server. If it becomes unavailable, requests are sent to another remote server. • **ROUND_ROBIN**: Switches to the next remote server for each new request thus attempting to share the load equally between all available remote servers.

RADIUS Attribute	Description
Remote Server List	The set of available remote RADIUS servers, each of which has the following attributes: • **Remote RADIUS Server Host Name**: The hostname or IP address of the remote RADIUS host • **Remote RADIUS Server Auth Port:** The UDP port number of the RADIUS authentication service (1812 standard, or 1645 default for RSA Authentication Manager) • **Response Timeout**: Defines how long to wait for a response from the server before possibly resending the request to the same or another server. Administrators can choose shorter timeouts on local servers than on remote servers. • **RADIUS Client Shared Secret:** The RADIUS client-server shared secret for encoding passwords in the request packets to the server.

6.2 Remote Web Service

The remote web service authenticator is similar to the remote RADIUS authenticator, in that it uses a remote third-party authentication server to validate OTPs. However, the remote calls are via HTTP rather than RADIUS. This allows the Echidna server to act as a proxy to a remote token authentication service.

The remote service is expected to be a legacy token or other OTP service.

The following table describes the attributes that administrators can change:

Web services attribute	**Description**
User ID Link Attribute	The user attribute(s) used to format the User-Name attribute in the remote web services request.
Passcode Length	The expected OTP length. If using PASSCODE_WITH_PASSWORD for the password combination method, the first passcode-length characters are the passcode. If using PASSWORD_WITH_PASSCODE then the last passcode-length characters are the passcode. Limits: 6, 7, or 8 characters
Passcode Ignore Characters	The characters to ignore in the passcodes that users enter. Anything matching this regular expression is removed from the OTP. Default: space ( ) and dash (-).
Generated ID length	The number of characters in the new identifier. This is used where the user attribute used to identify the OTPs is not already populated. If the value is 0, no ID is generated.
Password Combination Method	The way that the user-store password and the OTP are to be combined in the passwords submitted by users. This option can validate both the user-store password and the scheme-specific OTP. See Combining Echidna authentication methods with user store passwords on page 47 for more information.

Web services attribute	Description
Password Validation Order	The order in which the user-store password and the one-time passcode are validated: • **PASSWORD_THEN_PASSCODE**: With this option, a login challenge screen appears only if the submitted username and store password were correct. This may allow an attacker to try to guess user store passwords independently of the second factor (the remote authentication server's OTP). • **PASSCODE_THEN_PASSWORD**: The store password is validated only after the OTP is validated via the remote RADIUS server. The password submitted in the first call is encrypted and stored back into the challenge-state so it is available in the second call. With this option, the user store password is better protected from password guessing and denial of service. • **PASSWORD_OPAQUE**: This option is not relevant for Web Services.
Challenge Type	Specify whether the legacy token operates in challenge-response mode. If it does, it requires a challenge before the one-time passcode can be generated. In this situation, also specify how the challenge is generated. • **NO_CHALLENGE**: Either no challenge is required for the legacy token to generate the next OTP, or the RAS or NAS has already generated a challenge and submitted it with the user's response OTP. • **RANDOM_CHALLENGE**: The Echidna server generates a random challenge of some fixed length to be sent back to the user, assumed to be decimal digits only. • **SERVER_OBTAINED_CHALLENGE**: The remote authentication server must be called in order to obtain a valid challenge for the legacy token. • **SERVER_TRIGGERED_CHALLENGE**: The remote authentication server must be called to generate a challenge for the token, but the challenge value itself is communicated to the user through some other channel and not returned via Echidna.
Challenge Length	The number of digits that are generated in each challenge. This takes effect only if **RANDOM_CHALLENGE** is selected in **Challenge Type**.
Remote Server Bind Credential	The service account user name and password for invoking the remote authentication server.

Web services attribute	Description
Remote Generate Challenge Service Call	The remote HTTP service call definitions for a generate-challenge call (if needed). This call contains the following: • **Service URL**: The service URL that is invoked • **Method**: The service methods (HTTP POST or GET) • **HTTP Headers**: The HTTP headers to include • **HTTP Request parameters**: A template for the parameters or HTTP request body that is sent to the service point. • A set of patterns to recognise the response (and categorise as VALID/INVALID/TOKEN_UNAVAILABLE/SERVICE_UNAVAILABLE). Failure to connect or receive a response counts as SERVICE_UNAVAILABLE.
Generate-Challenge Responses	The set of patterns to categorise the response from the generate-challenge call.
Remote Verify OTP Response Service Call	The remote HTTP service call definitions for a verify-OTP call. This call contains the same parameters as **Remote Generate Challenge Service Call**.
Verify-Response Responses	The set of patterns to categorise the response from the verify-response call.

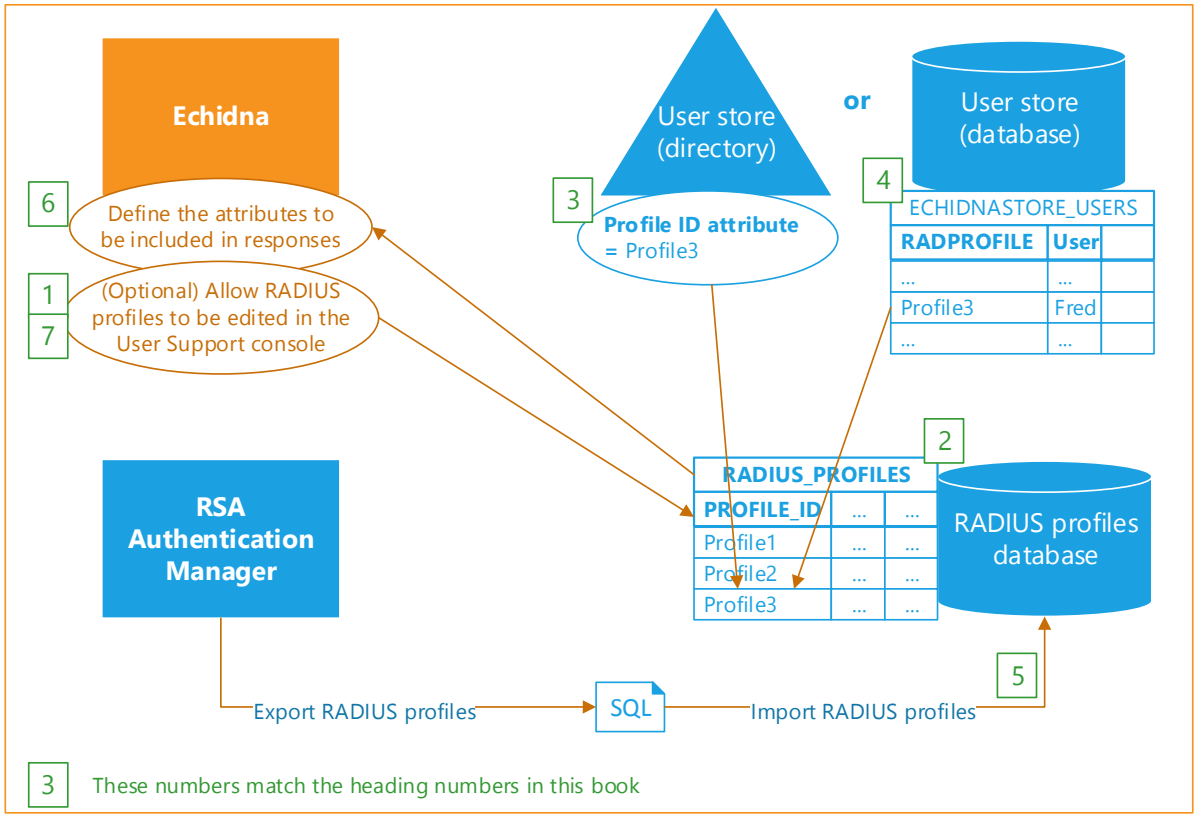
Web services attribute	Description
Trace	The file or database logging of the interactions with the remote server. This section include these parameters: • **History Buffer Size**: Specifies the number of trace records to keep in memory for viewing. Trace records are viewable via the Monitoring/Call Trace option. • **Current State Summary**: Controls the monitoring of recent call statistics for this remote server. Statistics are viewable via the Monitoring/Traffic option. • **Trace Log**: The location and format of the file-based trace log. The default log entry format is: **[{0}][{1,date,yyyy-MM-dd HH:mm:ss.SSS}][{2}][{3}][{4}][{5}][{6}][{7}]** Where the numbered parameters are provided by the fields of the java.util.logging.LogRecord entry being recorded. – {0} sequence number – {1} recorded when – {2} log level name – {3} source class name – {4} source method name – {5} escaped message – {6} exception class name – {7} escaped exception message text The message escaping can be done using CSV, JSON or XML escaping format to prevent any message delimiter ambiguity where a structured format is being used.
Persistent Log	The configuration of the database-stored audit log. See How persistence works on page 142.
Data Confidentiality Protector	The password protector to be used to protect the OTP-generating key material.

6.3 Migrate any RADIUS profiles

A RADIUS profile lets a RADIUS server include authorisation information in its responses to clients. This allows clients to rely on the server to provide authorisation information, which can be simpler than having each client work this out.

When migrating from a legacy RADIUS server, RADIUS profiles may already have been in use. Echidna can continue to support them. Echidna stores RADIUS profiles in a database, independent of which type of user store it uses.

The following diagram shows the components and connections described in this section:



To set up RADIUS profiles in Echidna, follow these steps:

1. Create a resolve process to manage the RADIUS profiles (see page 94)
2. Create a new database for the RADIUS profiles (see page 95)
3. Link the user to the RADIUS profile (for a directory user store only) (see page 96)
4. Link the user to the RADIUS profile (for a database user store only) (see page 97)
5. Populate the new RADIUS_PROFILES table (see page 98)
6. Include the RADIUS_PROFILES attributes in the authentication response (see page 99)
7. Allow the RADIUS profiles to be managed with the User Support console (see page 101)
8. Verify the RADIUS profile (see page 102)

6.3.1 Create a resolve process to manage the RADIUS profiles

A resolve process defines the user stores that can be managed in the User Support console. This is defined here so that it can be modified to omit the RADIUS profiles database in the next section. This prevents the RADIUS profiles database being misused as a user store.

1. Log in to the Administration console.
2. Click **CONFIG**, then click **User Resolution Procs**.
3. On the left, click **new** to create a new user resolution process.
4. Enter the following details:
   - **New user resolution process name**: support-manage-resolve, or a similar name that makes it clear that this resolve process controls which user stores can have content managed by the User Support console.
   - **User/Domain Patterns to Recognize**: Leave as is.
   - **Select user stores to resolve against**:
     - **Directory user stores**: Uncheck all directory user stores. These user stores cannot be managed in the User Support console.
     - **Database user stores**: Check the database user stores that will be manageable in the User Support console.
5. Click **Next**. Make no further changes to this process at this point.

6.3.2 Create a new database for the RADIUS profiles

1. Log in to the Administration console.
2. Click **CONFIG**, then click **User Stores**.
3. On the left, click **new** to create a new user store.
4. Enter the following details:
   - **New user store name**: db-profiles, or another name that makes it clear that this database contains only RADIUS profiles
   - **User store type**: JDBC Database with Existing Table
   - **Select the existing User-Resolve processes that should reference the new store**: Check only the resolve process that was defined in Create a resolve process to manage the RADIUS profiles on page 94. All other resolve processes should be unchecked.
5. Click **Next**, then set the following attributes of the database:
   - **Username Attribute**: PROFILE_ID
   - **Display Name Attribute**: PROFILE_LABEL. This attribute is used as the display name for RADIUS profiles appear for editing in the User Support console.
   - **Datastore Connection Name**: Use the same one used by the existing database user store if present. This enables foreign key constraints, which are used in a later section. By default, this is LocalStoreUnit.
6. Add a new table named **RADIUS_PROFILES** to the database:
   a. Find the **Tables** section, use the **Add** button  to add a table, then enter the following details:
      - **Schema Name**: RADIUS
      - **Table name**: RADIUS_PROFILES
   b. Set **Data Integrity Protector** to **utilMac**.
   c. Add the columns that are listed in this table:

Column Name	Column Type	Column Size	NOT NULL	Primary Key Order	Default Value
PROFILE_ID	VARCHAR	40	Checked	1	
PROFILE_LABEL	VARCHAR	140			
FRAMED_COMPRESSION	VARCHAR	40			
FRAMED_IP_ADDRESS	VARCHAR	40			
FRAMED_NETMASK	VARCHAR	40			
FRAMED_MTU	INTEGER				1500
FRAMED_PROTOCOL	VARCHAR	40			
FRAMED_ROUTING	VARCHAR	40			
FRAMED_ROUTE	VARCHAR	40			
SERVICE_TYPE	VARCHAR	40			

   d. Click **Create Persistence Table** to create the table.
7. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.

6.3.3 Link the user to the RADIUS profile (for a directory user store only)

Define an attribute in the user store to hold the RADIUS profile ID.

If the directory already has an unused attribute with a string syntax (such as **extensionAttribute1** for Active Directory), this can be used. If there are no appropriate attributes, define a new one.

1. Log in to the Administration console.
2. Click **CONFIG**, then click **Authenticators**.
3. In the **User Stores** list on the left, click the name of the directory.
4. Find the **Attribute Aliases** section, then add the following alias:
   - **Key**: profileID
   - **Value**: *attributename*, where this is the attribute that stores the RADIUS profile. For example, **extensionAttribute1**.
5. Click **Update**, then save the configuration changes.
6. Ensure that the user records contain the appropriate RADIUS profile identifier in the nominated attribute.

6.3.4 Link the user to the RADIUS profile (for a database user store only)

This section describes how to define a RADPROFILE column in the user store database and then link it to the PROFILE_ID column of the RADIUS_PROFILES table.

1. Log in to the Administration console.
2. Click **CONFIG**, then click **User Stores**.
3. In the **User Stores** list on the left, click on the user store name.
4. Find the **ECHIDNASTORE_USERS** table.
5. Add a column to this table, with the following details:
   - **Name**: RADPROFILE
   - **Column Type**: VARCHAR(40),
   - **Constraint**: FOREIGN KEY REFERENCES RADIUS.RADIUS_PROFILES(PROFILE_ID). This foreign key constrains the RADPROFILE value to contain only a value that is a valid PROFILE_ID from the RADIUS profiles table.

     A foreign key constraint can be set only if the RADIUS profile store and the user store use the same database connection. This was set in Step 5 in Create a new database for the RADIUS profiles on page 95.
6. Find the **Updates** section, then modify the definitions of the **users.create Update** and **users.update Update** to include the **RADPROFILE** column.
7. Click **Create Persistence Table** to create the table.
8. Find the **Attribute Aliases** section, then add the following alias:
   - **Key**: profileID
   - **Value**: RADPROFILE
9. Click **Update**, then save the configuration changes.

6.3.5 Populate the new RADIUS_PROFILES table

This section describes how to populate the RADIUS_PROFILES table directly, in bulk. Alternatively, the User Support console can be used to set up the RADIUS profiles one-by-one. See Allow the RADIUS profiles to be managed with the User Support console on page 101.

To populate the RADIUS_PROFILES table by executing a script directly against the database:

1. Export the current RADIUS profiles from RSA Authentication Manager to a text file.
2. Convert the exported text file into a script that can update the RADIUS_PROFILES database.

   For example, adapt the following example SQL script:

```
INSERT INTO RADIUS_PROFILES
  (PROFILE_ID, FRAMED_COMPRESSION, FRAMED_IP_ADDRESS, FRAMED_NETMASK, FRAMED_MTU,
FRAMED_PROTOCOL, FRAMED_ROUTING, FRAMED_ROUTE, SERVICE_TYPE)
VALUES
  ('PRIMUS','VJ-TCP-IP-header-
compression','10.20.15.228','255.255.255.255',1500,'PPP','None',NULL,'Framed'),
  (SECUNDUS','VJ-TCP-IP-header-
compression','255.255.255.1','255.255.255.255',1500,'PPP','None',NULL,'Framed'),
  ('TERTIUS','VJ-TCP-IP-header-
compression','10.20.15.254','255.255.255.255',1500,'PPP','Send-and-
Listen','10.20.200.0/24','Framed'),
```

3. (Optional) Add a human-readable name for each profile in the PROFILE_LABEL column.
4. Run the script.

6.3.6 Include the RADIUS_PROFILES attributes in the authentication response

Configure the authentication process to include the RADIUS_PROFILES attributes in the authentication response.

1. Log in to the Administration console.
2. Click **CONFIG**, then click **Authenticators**.
3. In the **Authentication Procs** list on the left, click **authenticationProc**.
4. Use the **Add** button  next to **Process Conditions** to create a new condition of the type **ADD**.
5. Scroll to the bottom of the page to see the newly added condition. For the remaining steps, this new condition is referred to as **condition.x**.
6. Add a sub-condition under condition.x, with the following details:
   - **New Condition Type**: EXACTMATCH
   - **Reference**: ${result.type}
   - **Match**: ACCEPT
7. Add another sub-condition under condition.x, with the following details:
   - **New Condition Type**: PRESENT
   - **Reference**: ${userContext.profileID}. This is the user attribute that is used to search for the RADIUS profile.
8. Add another sub-condition under condition.x, with the following details:
   - **New Action Type**: ExpandUserConStep
   - **Context Variable Name**: profile
   - **Datastore Connection Name**: Enter the same connection name that was entered in Step 5 of Create a new database for the RADIUS profiles on page 95.
   - **Data Confidentiality Protector**: utilProt
9. Add a query with the following details:
   - **Query Name**: profile.resolve
   - **Query SQL**: SELECT * FROM RADIUS.RADIUS_PROFILES WHERE (PROFILE_ID = ${userContext.profileID})

10. For each context variable to be set, add a sub-action under condition.x, using the following details:
    - **New Action Type**: SetVariableStep
    - **Var Name**: response.Framed-Compression
    - **Expression**: ${profile.FRAMED_COMPRESSION}

    Repeat this step for each profile attribute that needs to be included in the response:

Var Name	Expression
response.Framed-Compression	${profile.FRAMED_COMPRESSION}
response.Framed-IP-Address	${profile.FRAMED_IP_ADDRESS}
response.Framed-IP-Netmask	${profile.FRAMED_NETMASK}
response.Framed-MTU	${profile.FRAMED_MTU}
response.Framed-Protocol	${profile.FRAMED_PROTOCOL}
response.Framed-Routing	${profile.FRAMED_ROUTING}
response.Framed-Route	${profile.FRAMED_ROUTE}
response.Service-Type	${profile.SERVICE_TYPE}

11. Click **Update**, then save the configuration changes.

6.3.7 Allow the RADIUS profiles to be managed with the User Support console

The User Support console is a web application that comes with Echidna. It lets customer support staff edit user and token details. It can also be used to update RADIUS profiles.

If the RADIUS profiles can be managed by the User Support console, service staff can create, edit, and delete RADIUS profiles using the User Support console.

If the RADIUS profiles cannot be managed in the User Support console, the only way to edit them is by editing the database directly.

To allow RADIUS profiles to be managed in the User Support console, expose a **user registration** web service so that the table can be managed through the User Support web application. A user registration service allows the entries in any database-backed user stores referenced in a given **user resolution** process to be managed.

1. Log in to the Administration console.
2. Click **CONFIG**, then click C**onnectors**.
3. In the **Services** list on the left, click **webServices**.
4. Find the **User Reg Services** section, use the **Add** button  to add a service, then enter the following information:
   - **Name**: userReg
   - **User Resolve Process Ref**: support-manage-resolve
5. In the **Published Services** list, find the new **userReg** row, then check the **Service Enabled** box.
6. Click **Update**, then save the configuration changes.

The RADIUS profiles can now be edited in the User Support console. To edit a RADIUS profile in the User Support console, follow these steps:

1. Log in to the User Support console.
2. Click **Manage Users**, then select **Users** under the **db-profiles** user store.
3. Click **New** to create a new profile. Each field of the RADIUS_PROFILES table is shown.
4. Enter values for each field that is required, then click **Create**.

6.3.8 Verify the RADIUS profile

Use a RADIUS profile testing tool to verify the configuration changes. RADIUS testing tools are available online.

To test the RADIUS profile changes, send an authenticate request and check the response. The response should include the correct RADIUS_PROFILES attributes based on the RADPROFILE value assigned to the user.

The following is a sample authentication response:


```
"authenticateResponse":{
"method":"authenticate",
"result":{
"@name":"SUCCESS",
"@type":"ACCEPT",
"@packetIdentifier":276,
"userIdent":{
"@userID":"User101",
"@domain":"dbMySQL"},
"extra":[
{
"@key":"Framed-MTU",
"@value":"1500"},
{
"@key":"Framed-IP-Netmask",
"@value":"/255.255.255.255"},
{
"@key":"Salt-User-Domain",
"@value":"dbMySQL"},
{
"@key":"Service-Type",
"@value":"Framed(2)"},
{
"@key":"Salt-User-Name",
"@value":"User101"},
{
"@key":"Framed-IP-Address",
"@value":"/255.255.255.1"},
{
"@key":"Framed-Protocol",
"@value":"PPP(1)"},
{
"@key":"Framed-Compression",
"@value":"VJ-TCP-IP-header-compression(1)"},
{
"@key":"Framed-Routing",
"@value":"None(0)"}]}
```

Chapter 7: Configure services and access points

Use this chapter to configure Echidna to accept requests from clients, using the configuration components shown in this diagram:

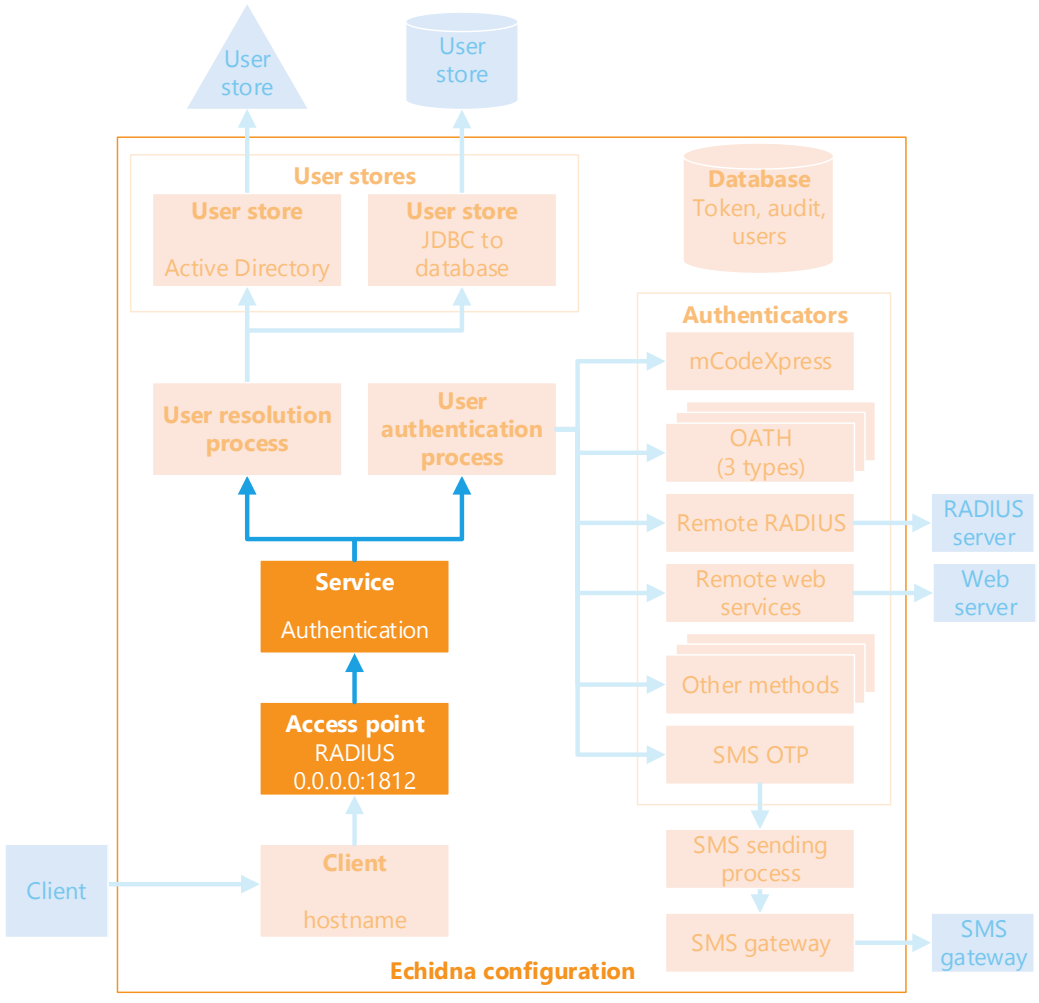


Figure 7: The configuration that handles services and access points

7.1 How services and access points work

An *access point* is a port on which Echidna listens using a particular protocol. Echidna can have multiple access points that use the same protocol, each with different settings.

A *service* configuration controls the details of a service that may be published by a particular access point. The same service may be published on multiple access points.

Echidna can have many access points, for many purposes. This section discusses access points that allow clients to connect to Echidna. For information about access points that support monitoring, see Monitor Echidna on page 119.

After following the **Setup** wizard as described in the Installation Guide, Echidna has the following services:

- A service for web services requests (HTTP or HTTPS)
- A service for RADIUS requests

7.1.1 How auto-startup works

If set to auto-startup, an access point starts its services and listens on the specified socket after the application server (Tomcat) and Echidna have started.

When an access point is stopped, any linked services are also stopped. Multiple access points for a single service stop the underlying service only when the last access point is stopped.

The Echidna engine starts automatically when Tomcat is started only if **Enable Auto-Start on Reboot** option was left checked in the first Setup wizard screen. Otherwise the services can start up only after an administrator logs in to the Administration console.

7.2 Configure services

A service configuration controls the details of a service that may be published by a particular access point. The same service may be published on multiple access points.

1. Log in to the Administration console.
2. Click the **Connectors** tab.
3. In the **Services** section on the left, click **new** and then select a service type:

Service type	Description	Components that can be edited
RADIUS Authentication	Publishes on RADIUS Authentication access points	• The **user resolution** process used to look up the user in the user store(s) during authentication. • The **user authentication** process used to determine which authenticators to apply to the resolved user. • The name of a heartbeat user to recognise as being used for RADIUS monitoring, for which ACCESS-CHALLENGE is always returned. • The trace log and monitoring configuration for the service.
Web Services	Authentication, token, provisioning and monitoring services for publishing on TCP/HTTP web service access points	• A set of user-authentication services, each one having the equivalent configuration of a single RadiusServiceConfig (above). • Whether to enable the Traffic Summary Service. • Whether to enable the Trace Log Service. • The set of token authentication services that this web service publishes.
JMX Monitoring	Publishes JMX management beans on a JMX TCP access point	• The SSL context to use if the JMX service is to use SSL. • The optional environment settings used in creating the JMXConnectorServer. • Whether to expose the platform MBeans.

Service type	Description	Components that can be edited
Derby Network Server	Publishes an embedded Apache Derby database instance on a Derby Network access point	• Maximum JDBC Client Threads. This is the maximum number of threads that are used for JDBC client connections in the Derby Network Server. If <= 0, connection threads are created when there are no free connection threads. • JDBC Thread Time Slice (milliseconds). Number of milliseconds given to each session before yielding to another session, if <=0, never yield. This should be set and is only relevant if maxThreads > 0. • Optional environment settings used in creating the Derby network server controller. • The JDBC connection of the embedded Apache Derby DB to be published. Note: To use this service, the derbynet-10.8.2.2.jar library must be installed in the /usr/share/tomcat/lib folder on the appliance.

7.3 Configure access points

7.3.1 Add an access point for client requests

1. Log in to the Administration console.
2. Click the **Connectors** tab.
3. In the **Access Points** section on the left, click **new**.
4. Select the access point type:
   - **RADIUS Authentication**: RADIUS protocol (listening on a UDP port). Note: use of an IPv6 format address with UDP may require a Java 7+ runtime.
   - **TCP (HTTP) Authentication Services**: Web services over HTTP.
   - **TCP (HTTPS) Authentication Services**: Web services over HTTPS.
   - **TCP Authentication Services via Admin Web Server**: Web services interface to be exposed via the administration web server's own HTTPS interface (rather than a separate socket being opened).

   For the following access point types, see Monitor Echidna on page 119:
   - **JMX Remote Monitoring Services**: Publishes the JMX management beans with authentication using the RADIUS client names and shared secrets.
   - **Derby Network Access Service**: Publishes the embedded Derby database over a network listener, with authentication using the RADIUS client names and shared secrets.
   - **Shared State Subscriber**: Allows Echidna server peers in high availability configurations to securely share state information about timed lockouts and generated SMS OTP codes.
5. Select the address and port for the new access point.
6. Select the service that will be associated with this access point.
7. Click **Next**. The access point is created with default settings.
8. Edit the details of the new access point:

Access point parameter	Description
Host	Host IP address to listen to. Use **0.0.0.0** for all available IPv4 addresses, **::** for all available IPv6 address, or a specific address to restrict the listener to a single network interface.
Port	Port number to be used for server socket
SO Timeout (Seconds)	Socket timeout
Published Service	Name of the existing service that this listener is publishing
Startup Type	Start-up type of the access point (MANUAL or AUTO)

Access point parameter	Description
Allow Unauthenticated Access	Checked: Allow access without username/password or other explicit credentials, but infer the client name from the source IP address. Not supported for RADIUS listeners.
Allow Selected Clients Only	Checked: Only permit the explicitly selected clients to access the service via this listener. Unchecked: Allow any configured clients.
Selected Clients	Restricted list of clients allowed to invoke this service.
Add New	Add a new reference to a RadiusClientConfig

9. Click **Update**.
10. Start the access point listener.
11. Save configuration changes (see page 128)

7.3.2 Edit an access point

The access point configuration is only editable when the listener is not running.

1. Log in to the Administration console.
2. Click the **Connectors** tab.
3. In the **Access Points** section on the left, click the name of the access point.
4. At the bottom of the page, click **Stop**.
5. Edit any of the items listed in Step 7 above.
6. Click **Update**.
7. Start the access point listener.
8. Save configuration changes (see page 128)

Chapter 8: Configure clients

This chapter describes how to configure Echidna to authenticate users on behalf of clients, as shown in this diagram:

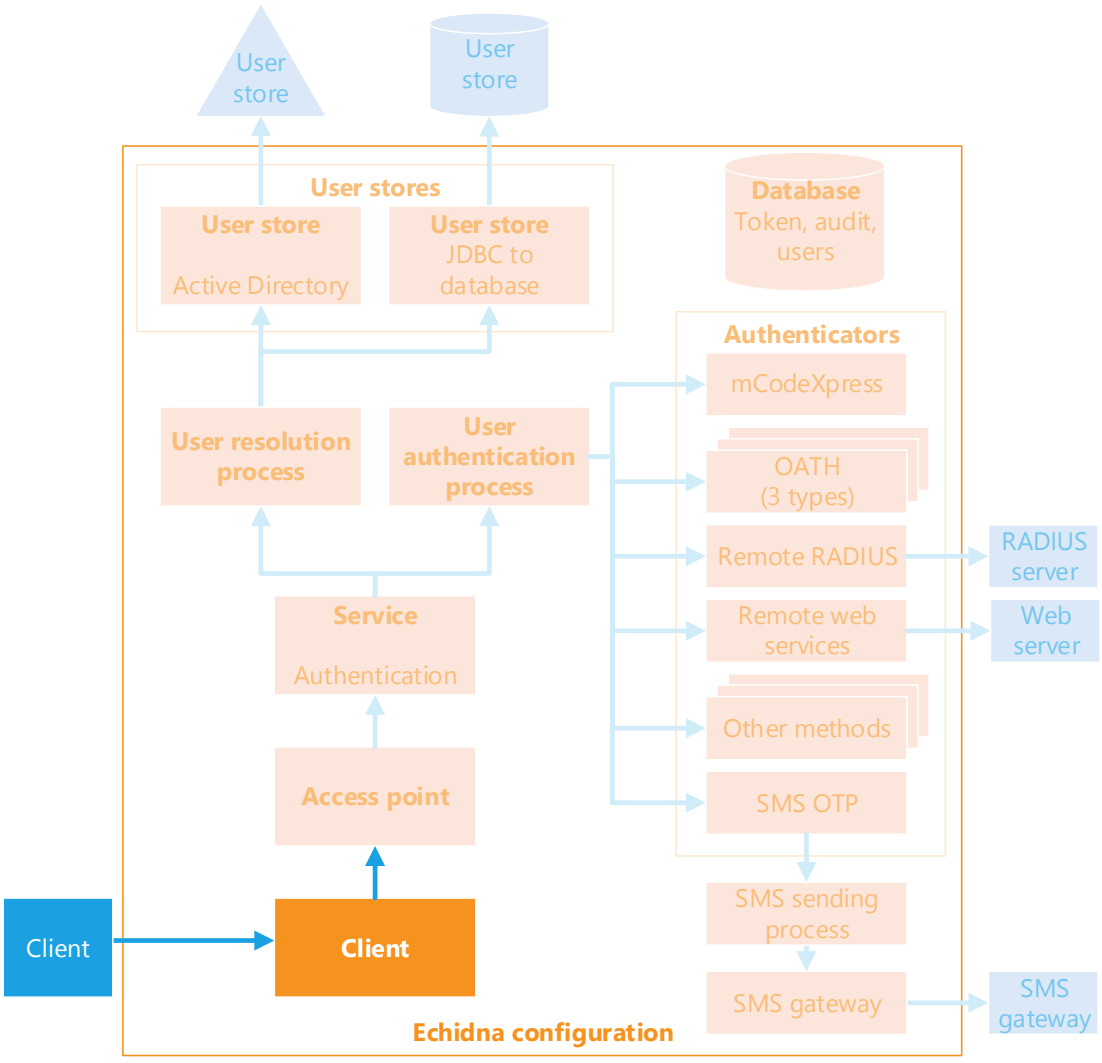


Figure 8: The configuration that deals with clients that can connect to Echidna

8.1 Allow clients to invoke Echidna

Echidna clients are other services or applications that communicate with Echidna via RADIUS, web services, or any of the other access point protocols described above. For RADIUS, the client must be identified using the host name or IP address of the server making the connection.

To allow a client to use Echidna for user authentication, the client needs to be added to the configuration of Echidna so that access-request packets from it are processed by Echidna.

8.1.1 Add a client to Echidna

1. Log in to the Administration console.
2. Click the **Connectors** tab.
3. In the **Clients** section on the left, click **new**.
4. Enter the host name of the Echidna client which can be resolved into an IP address.
5. Click **Next**.
6. Update any of the following:
   - Name of the Echidna client
   - Net mask (blank by default) for the Echidna client
   - Shared secret to be used with the Echidna client
   - View the interface name, canonical name, and IP address of the Echidna client
7. Click **Update** to save the changes.

Chapter 9: Configure passwords, keystores, and SSL

This section describes how Echidna secures sensitive information, negotiates SSL connections, and uses the Java environment's cryptographic services for other tasks:

Echidna protects the passwords and other information that it manages by encrypting or signing the relevant data using cryptographic keys and services. These services are configured in the **Crypto** section of the Administration console. The elements most commonly referenced from the rest of the Echidna configuration are the Password Protectors and the SSL Contexts, but to provide those services, the underlying cryptographic key stores also need to be configured.

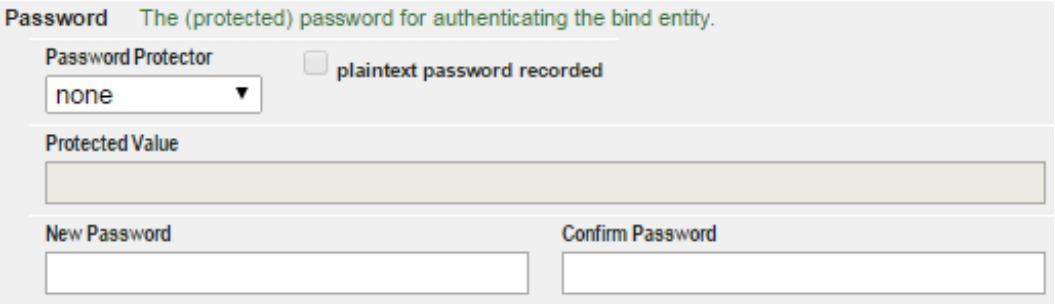
9.1 Password protectors and protected passwords

Echidna protects the passwords and other sensitive information that it stores and manages.

A password protector in Echidna is a wrapper around a protected cryptographic key. They are named **password protectors** because they are commonly used to encrypt and recover sensitive values within the Echidna configuration or databases. However, a password protector can also sign or MAC data and manage RSA key pairs.

A password protector is used for encrypting and decrypting passwords managed by Echidna, such as those used to access the server configuration or an external user store. The default password protection method uses an identified key from a configured keystore to perform this encryption and decryption.

A protected password is used in other Echidna configuration elements where a sensitive value (a password, shared secret, or key) needs to be recorded, as in the following screenshot:



If **none** is selected, the password in the New Password box is stored in plaintext, without encryption. When a password is stored in this way, the **plaintext password recorded** box is automatically checked.

If a password protector is selected, the password that is entered in the New Password box is protected using that protector. The encrypted value is shown in the **Protected Value** box.

9.1.1 Add a new password protector

A default Echidna configuration has the following password protectors:

- **Utility Password Protector (utilProt)**: The default AES-256 secret key used to encrypt and recover the shared secrets used with RADIUS clients, LDAP user stores, bootstrap store users' passwords, HTTP SMS gateways, and Salt mCodeXpress cryptographic keys.
- **Utility MAC Generator (utilMac)**: The default HMAC key used to calculate TEMAC signatures on data sets persisted to a database table.
- **Server Identity Protector (serverIdentity)**: The default RSA private key and certificate used for server identify on published services over TLS, for mutual-SSL authentication on outgoing connections, and for wrapped token data imports.
- **Operator Password Protector (operatorProt)**: An internal AES-256 secret key unlocked by the operator password and used to encrypt and recover the keystore passwords needed to unlock the other password protectors.

If the default protectors are already in use and some alternative protector with a different algorithm, key size, or cryptographic provider is required, new protectors (keys) can be configured and generated as follows:

1. Log in to the Administration console.
2. Click the **Crypto** tab.
3. In the **Password Protectors** section on the left, click **new**.
4. Type a name for the new password protector, then click **Next**.
   The new password protector is created, with default settings.
5. Update any of the following settings:

Password protector parameter	Description
Key store	The cryptographic keystore that holds this protector's secret key or private key and certificate chain (see Manage keystores on page 115).
Key Password Specified	Most keystores require a store password to unlock access to the store contents. Some types of keystore (including JKS and JCEKS) also allow or require each key entry within the store to have its own key password, so both the store password and the key password are needed to gain access to the protected key functions. If the Key Password Specified checkbox is not selected, the key is protected with the store password only. If the checkbox is selected, the additional configuration items allow the protected key password to be configured.
Key Alias	The key alias is the label for this protector's secret or private key entry within the cryptographic keystore.

Password protector parameter	Description
Subject DN	Subject name for certificate generation (only for RSA keys). If there is no private key entry with the given alias in the keystore, a new private key and self-signed certificate can be generated using this as the subject distinguished name.
Auto Generate When Missing	The password protector consists of a key and a reference to a keystore. • **Checked**: If there is no private or secret key entry with the given alias in the keystore, Echidna attempts to generate one and store it into the keystore at the time when it is first needed. This might be the case for a newly configured stand-alone server where there is no need to be sharing the same key value across multiple servers. • **Unchecked**: During migration or in cases where either the keystore or the key is not present, leaving **Auto Generate When Missing** unchecked causes an error if an attempt is made to use the protector. In a single-server environment, either check this box or pre-generate the key (using the **Generate** button which appears when the key doesn't yet exist).
Algorithm	This is the cryptographic algorithm of the protector's private or secret key. The possible values depend on the available Cryptographic Providers in the Java runtime environment, but the standard names are given in the KeyGenerator (for secret keys) and KeyPairGenerator (for private keys) sections of the Java™ Cryptography Architecture Standard Algorithm Name Documentation page at http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html. The common values for Echidna are AES for encryption keys, HmacSHA1 and HmacSHA256 for MACing keys, and RSA for identity keys.
Key Size	The key size in bits to use when generating a new key. Typically 256 for AES, 1024 or 2048 for RSA, 160 for HmacSHA1 and 256 for HmacSHA256.
Padding Type	When the key is used for encryption, this is the type of padding to apply. Typically PKCS5Padding for symmetric keys (AES, DES) and for RSA one of OAEPPadding, PKCS1Padding or PKCS5Padding.
Cipher Provider	Leave blank to pick up the platform default cryptographic provider, or override with a provider name to control which cryptographic provider implements the encryption and decryption. For a list of the available providers in Echidna, go to the Config > License > Diagnostics > Crypto Providers page. Typical provider names are SUN, SunRsaSign, SunEC, SunJSSE, SunJCE, SunJGSS, SunSASL, XMLDSig, SunPCSC, SunMSCAPI, SunPKCS11-NSS and BC. For a description of the SUN providers, see http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html Additional providers can be loaded in the Config > Crypto > Providers section.

6. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
7. Save configuration changes (see page 128).

9.2 Manage keystores

Keystores are containers for the secret and private keys and certificates. Password protectors use keystores to encrypt sensitive data involved in the operation of Echidna.

The default Echidna configuration has four keystores:

- **Operator Keystore (operatorKs)**: JKS keystore for the Operator password protector
- **Utility Keystore (utilKs)**: JCEKS keystore for the utilProt and utilMAC password protectors
- **Server Keystore (serverKs)**: JKS keystore for the Server Identity Protector
- **Trust Keystore (trustKs)**: JKS keystore that holds known trusted certificates used by the SSL context

To add a keystore to Echidna, follow these steps:

1. Log in to the Administration console.
2. Click the **Crypto** tab.
3. In the **Keystores** section on the left, click **new**.
4. Type a name for the new keystore, then select the type of keystore:
   - **JCEKS**: Supports symmetric secret keys
   - **JKS**: Supports trusted certificates and private keys

     To use different keystore type (PKCS12 or CaseExactJKS), do not select either type, but update the keystore type on the next screen.
5. Click **Next**.
6. Update any of the following settings:

Keystore parameter	Description
Keystore Type	The keystore type. By default, use JCEKS for stores containing symmetric keys (AES, DES, HmacSHA256) and JKS for stores containing private keys and certificates. For more information, see List the crypto providers supported by Echidna on page 116.
Keystore Provider	The JCE provider name, if the keystore instance should be limited to a given provider. For more information, see List the crypto providers supported by Echidna on page 116.
Keystore Path	If the keystore is stored in the filesystem, enter the path to the keystore file (relative to the config XML file).
Keystore Bytes	If Keystore Path is empty, the keystore bytes can be stored directly in the config file instead (base64 encoded). If both are left blank, a new empty keystore is created, and the first time it is updated (a key is generated for it or a certificate is imported) the keystore bytes is populated.

Keystore parameter	Description
Store Password	Most keystores require a store password to unlock access to the store contents. This group of properties is the protected store password (as described in protected password description above). It allows the password to be provided and optionally encrypted by a password protector (just not one that itself uses a key from this keystore). If a password protector is not used, the password is recorded in plaintext in the config.xml file, but the password value is still not exposed in the web page (the **plaintext password recorded** checkbox is selected instead).

7. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
8. Save configuration changes (see page 128).

9.2.1 List the crypto providers supported by Echidna

1. Log in to the Administration console.
2. Click the **License** tab.
3. In the **Diagnostics** section on the left, click **Crypto Providers**.
   A list of the supported providers appears on the right.
   For supported keystore types, look for a **Services** section, then find the lines that start with the **KeyStore** typename.

9.3 Configure SSL

Echidna can act as an SSL client when connecting to external user stores or SMS gateways. It can also act as an SSL server to provide access to the Administration console and the User Support console.

When the Echidna appliance starts for the first time, it generates an SSL certificate and a private key for use by the Apache Tomcat server. Echidna uses this self-generated certificate for all Tomcat HTTPS connectors. After deployment, the administrator should update this certificate. For instructions, see "Update the SSL certificate" in the *Echidna Installation Guide*.

To add an SSL context to Echidna, follow these steps:

1. Log in to the Administration console.
2. Click the **Crypto** tab.
3. In the **SSL Contexts** section on the left, click **new**.
4. Enter a name for the new context and click **Next**.
5. Update any of the following settings:

SSL context parameter	Description
Protocol	The protocol used to create the SSL Context instance. Default is TLS. Other standard names are SSL, SSLv2, SSLv3, TLSv1, TLSv1.1 and TLSv1.2. It is not recommended to configure any protocol older than SSLv3.
Provider	Specify this to control which Java crypto provider's implementation of the SSL Context protocol to use.
Want Client Auth	Check this box if this SSL context is to be used as for a server socket, and the server should prompt the client for an SSL certificate, but not fail if the client does not present one.
Need Client Auth	Check this box if this SSL context is to be used as for a server socket, and the server should require mutual authentication. In this situation, the client must use its own SSL client key and certificate during the connection establishment.
Priv Key Name	The name of the password protector that holds the private key and certificate to use in this context.
Trust Store Name	The name of the keystore holding the trusted certificate collection to use in this context. This is likely to be **trustKs**.

SSL context parameter	Description
Host Verifiers	When making an HTTPS connection to a remote server, if the hostname from the service URL doesn't match the contents of the returned SSL server certificate, an exception is normally generated to indicate the server is not trusted (even if the certificate itself is trusted). By adding entries to this Host Verifiers table, the acceptable subject CN or DN values can be configured for each hostname. This may be necessary if an IP address is used in the service URL (perhaps because no DNS resolution is 8available for the actual hostname).
Supported SSL Protocols	The list of SSL context protocol names supported by this SSL context. The list is obtained from the loaded provider class at runtime. The values can be used for setting the Protocol parameter.
Supported Cipher Suites	The list of cipher suites that the SSL context provider supports. The list is obtained from the loaded provider class at runtime. The values can be used for setting the names in the Enabled Cipher Suites list.
Last Trust Trace	When a user tests the connection to an LDAPS user store, the certificate trust chain processing of the SSL hand-shake is captured and can be found here.

6. At the bottom of the page, click **Update**. Echidna immediately uses the new configuration.
7. Save configuration changes (see page 128).

Chapter 10: Monitor Echidna

Echidna provides easy-to-use facilities for Echidna operators such as network and security administrators to monitor its activities, including RADIUS packet tracing, SMS gateway tracing, and traffic monitoring.

Administrators can monitor the following aspects of Echidna:

- Incoming authentication requests
- Outgoing traffic to SMS gateways
- Calls to remote RADIUS servers, where Echidna proxies to a legacy authentication server such as RSA Authentication Server
- Use RADIUS or web services to check Echidna health using a load balancer
- Monitor using JMX. Turn this on in Connector, Services. It requires a **JMX Remote Monitoring Services** access point.
- Monitor the internal Derby database. Turn this on in Connector, Services. It requires a **Derby Network Access Service** access point.

10.1 Enable Echidna monitoring

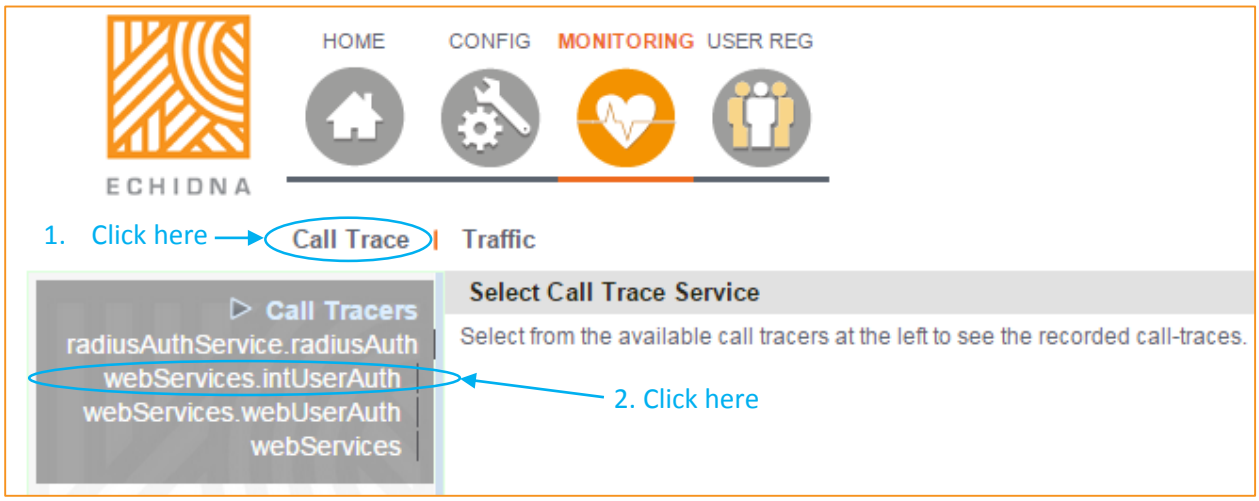
To enable Call Trace and Traffic monitoring from within Echidna:

1. Edit the web service from within Echidna/Connectors/Services
2. Under each Trace section enter 200 (or similar) into the History Buffer Size field.

It's also a good idea to deselect **Tracing Enabled** for **trafficSum** and **TraceLogs**, otherwise logging gets recursive and noisy.

10.2 Monitor incoming authentication requests

Access point monitoring allows Echidna operators to view the recent RADIUS traffic between Echidna and its clients. This information can be accessed in the Call Trace tab.



Information about the packets including the client IP addresses, packet type, and time sent or received is shown.

10.3 Monitor outgoing traffic to SMS gateways

SMS Gateway monitoring allows Echidna operators to view the usage statistics of the SMS gateways. This information can be accessed in the SMS Trace tab. This tab is not available if no SMS gateway is configured.

Information about recent SMS messages sent for Echidna can also be viewed by clicking on the number next to **DAYCOUNT**.

Traffic monitoring allows Echidna operators to view the volume of traffic that has passed through Echidna. This information can be accessed in the Traffic tab.

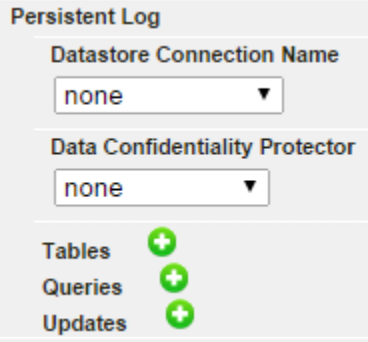
Usage statistics including the packet count, start delay, and finish delay is shown.

10.4 Write audit information to the database

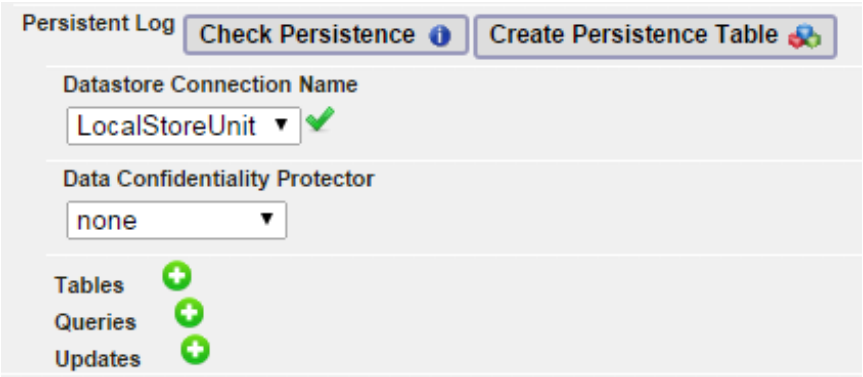
Administrators can enable auditing for many areas of Echidna, including the following:

- User stores
- Services
- Trace configurations
- Authenticators

In the configuration for each of these areas of Echidna, look for the Persistent Log section:



1. Select the **Datastore Connection Name** to be used for the Persistent Log.
2. Click **Update** to save the change.
3. Use the Tables section to create the database table that is used for the persistent log:



4. For the persistent log section in the RADIUS services and web services pages, there is also a **Define Default Audit Log Table** button. This inserts a definition for a default **ACCESS_REQUEST_AUDIT** table including the SQL Update statement needed to append to it, and some SQL Queries to interrogate it. The table definition and SQL elements can be further refined.
5. Click **Create Persistence Table** to apply the definition.
6. (Optional) Use **Data Confidentiality Protector** to set a protector on this data. The default protector is **utilProt**.

   Data Integrity Protection can also be set to enable tamper-evident logging. This uses utilMac to generate a MAC for each row, and this is stored in the TEMAC column. A check mark under the TEMAC column indicates that any value in this column is ignored when generating the MAC.

10.5 Messages that can appear in Echidna

This section lists messages generated by Echidna. These messages can be returned in service calls, appear in audit logs, or sent in SMS messages.

All of these messages are controlled by named message formats. These formats can be reviewed and configured on the Config > Msg Specs page.

The same page also lists the possible named results for authentication requests in the current configuration. Each named result has an associated log-message format (to provide the result summary in the Echidna event logs) and a reply-message format (to provide the reply-message back to the caller as part of the authentication result).

The common result specifications are listed in the table below.

Name	Type	Description
CHALLENGE_ALTAUTH_RES	CHALLENGE	The initially selected authentication mechanism for the user is not available (due to a service or token-unavailable response), but a fall-back has been configured. The user is being challenged to enter the credentials appropriate to the fall-back mechanism.
CRED_VALIDATION_FAILED	REJECT	Validation of the user credential failed due to some kind of service error. The validity of the credential cannot be established.
GENTOK_BAD_FORMAT_RES	REJECT	The submitted OATH OTP value was not in the expected format (6 decimal digits)
GENTOK_OTP_CHAL_RES	CHALLENGE	The userID and password have been collected but the user must now submit the OTP from the OATH token.
GENTOK_WRONG_OTP_RES	REJECT	The submitted OATH OTP was incorrect or outside the allowed time window.
HEARTBEAT_USER_RES	CHALLENGE	The result recorded when an access-request is received for a heart-beat monitoring user.
IGNORE_BADMSGAUTH	DISCARD	The validation of the message-authenticator in the access-request failed. The packet is ignored.
IGNORE_DUPLICATE	DISCARD	The received RADIUS request packet was a duplicate of a previously seen packet that is already being processed. This can be due to a timeout-retry mechanism at the client. This packet is not processed.
IGNORE_INVALID_CLIENT	DISCARD	The RADIUS access-request appears to originate from a client address that has not been configured as a RADIUS client. The packet is ignored.

Name	Type	Description
IGNORE_MALFORMED	DISCARD	The received RADIUS request packet was malformed in some way and is ignored in conformance with the RADIUS RFC 2865 rules.
IGNORE_PACKET_TYPE	DISCARD	The RADIUS request packet was not of the type (Access-Request) supported by this listener. The packet is ignored.
INVALID_AUTHSTATE	REJECT	The RADIUS state attribute included in the access-request was not a valid state that was created by this RADIUS server.
INVALID_CONFIG	REJECT	Some part of the configuration at the server appears to be invalid or incomplete and has caused a failure in processing the request.
INVALID_CREDS	REJECT	The access-request provided invalid user credentials (password).
INVALID_REQUEST	REJECT	The request was invalid due to a problem decoding the password attribute. The client's RADIUS shared-secret may be incorrect.
INVALID_USER	REJECT	The access-request specified an invalid user name.
LOCKEDOUT_USER	REJECT	The timed user-lockout mechanism has been triggered for the user name specified in the access request. The user must wait for the lockout to expire before trying to log in again.
MCODEX_BAD_FORMAT_RES	REJECT	The submitted OTP had an invalid length or format (expected 8 decimal digits).
MCODEX_OTP_CHAL_RES	CHALLENGE	The user-store password has been collected, the user should now be prompted for the next mCodeXpress one-time passcode.
MCODEX_WRONG_OTP_RES	REJECT	The submitted mCodeXpress one-time passcode was incorrect or outside the acceptable time window.
NOT_ENABLED_RES	REJECT	The registered OTP token for the user has been disabled or locked.
NOT_REGISTERED_RES	REJECT	There is no registered OTP token associated with the user.
NO_AUTHMECH_SELECTED	REJECT	No authentication mechanism was selected, since for all the configured mechanisms the pre-conditions for application to the specific user were not met.

Name	Type	Description
OATH_BAD_FORMAT_RES	REJECT	The submitted OATH HOTP had an invalid length or format, or it had a KCV prefix, and the KCV didn't match the recorded key id.
OATH_OTP_CHAL_RES	CHALLENGE	The user-store password has been collected, the user should now be prompted for the next OATH one-time passcode.
OATH_WRONG_OTP_RES	REJECT	The submitted OATH one-time passcode was incorrect or outside the time window.
OTP_CHAL_RES	CHALLENGE	The user-store password has been collected, the user should now be prompted for the one-time passcode.
PASSWORD_NOT_SET	REJECT	No password has been set for the user in the user store, so the access-request password cannot be validated.
REJECT_MALFORMED	REJECT	The access-request packet was malformed in some way that does not require it to be ignored. An explicit access-reject is returned.
RR_ACCEPT_RES	ACCEPT	The remote RADIUS server has returned an access-accept, and the local authentication process has completed successfully.
RR_CHALLENGE_LOCAL_RES	CHALLENGE	The user-store password has been collected and now the user should be challenged for the remote RADIUS server password. No remote RADIUS call has been made yet.
RR_CHALLENGE_NOMSG_RES	CHALLENGE	The remote RADIUS server returned an access-challenge but with no specific reply-message. The challenge with a generic reply-message should be passed through back to the caller.
RR_CHALLENGE_RES	CHALLENGE	The remote RADIUS server returned an access-challenge including a reply-message which should be passed through back to the caller.
RR_ERROR_RES	REJECT	There was a problem invoking the remote RADIUS server, or the user has not been properly configured for it (such as missing a remote token-identifier).
RR_REJECT_RES	REJECT	The remote RADIUS server has returned an access-reject.
RR_TIMEOUT_RES	REJECT	The timeout has expired waiting for the remote RADIUS server to respond.
SERVICE_ERROR_RES	REJECT	The remote authentication service is failing with some specific error message.

Name	Type	Description
SERVICE_UNAVAILABLE_RES	REJECT	The remote authentication service (RADIUS or Web Service) is not responding as expected, making the service unavailable.
SUCCESS	ACCEPT	Generic authentication-success result
TOKEN_UNAVAILABLE_RES	REJECT	The specific token identified for the resolved user does not exist in the remote service or has been disabled.
USER_MISSING_ATTR	REJECT	There is a user attribute missing from the user's context (the user-store entry) that is required for the selected authentication mechanism. For example, the mobile phone number is missing for SMS authentication.
USER_RESOLVE_FAILED	REJECT	Resolving the user in a user store failed due to some kind of service exception, such as the directory or database not being available.
USER_UNAUTHORIZED_RES	REJECT	Some group or role membership is required to allow the user to authenticate, and they do not meet the required condition.
WRONG_CHALRESPONSE_RES	REJECT	The submitted OTP was incorrect, but the lock-threshold has not been exceeded yet, so it is possible for the user to try again immediately.

Chapter 11: Register users for authentication

Before Echidna can support an end user authenticating with one of the configured authentication methods, the user record must be available and the relevant token may have to be assigned or registered.

This chapter describes how to register users via the Administration console. For information about registering users via the User Support console, see the *Operations Guide*.

In a production system, users are usually registered by support staff via the User Support console. However, it is useful to be able to set up some users while deploying and testing Echidna, before the roles for support staff have been configured.

When an organisation has an existing user store, it usually has a process for user management. If that is not available, the User Support console can be configured to manage the user store. The User Support console is also the recommended way to assign and register tokens.

The Administration console lets administrators do only some user registration and token management tasks:

- Manage user records in database and internal user stores, but not in external LDAP or Active Directory user stores. The Administration console can handle the following user tasks:
  - Add user records
  - Delete user records
  - Edit user records (limited)
- Register Salt mCodeXpress tokens and limited-use-temporary passwords, but no other types of tokens. The Administration console can handle the following token tasks:
  - List existing registrations
  - Register a token on behalf of a user
  - Delete an existing registration

11.1 Define a user who will use a Salt mCodeXpress token

1. Log in to the Administration console.
2. Click the **User Reg** button at the top.
3. Click the **mCodeXpress** tab.
4. On the left, click **Register User**.
5. Enter the user's details:
   - **RADIUS client**: The client that the user is authenticating from.
   - **Domain**: The user store that contains this user's record.
   - **Username**
6. Click **Find User**.
   If the user is found in the user store, the **Register mCodeXpress Token** page appears.
7. Enter the details of the user's Salt mCodeXpress token, then click **Register Token**.

11.2 Disable or delete a token

To disable or delete a token, the administrator only has to enter the mCodeXpress tab, locate the token to be disabled or deleted in the list of tokens displayed, and click on the correct button under the Action column.

11.3 Register a new Salt mCodeXpress token

To register a new token on behalf of a user:

1. Log in to the Administration console.
2. Click the **User Reg** button at the top.
3. Click the **mCodeXpress** tab.
4. On the left, click **Register User**.
5. Select the RADIUS client and enter the user's domain and user name, and then click **Find User**.
6. On the phone, get the registration information from the Salt mCodeXpress app:
   a. Open the Salt mCodeXpress app.
   b. Ensure that the phone has network access, then tap **Activate**:
   c. Enter a new 6-digit pin, then tap OK. The app shows three new codes.
7. Back in the Administration console, type in the three codes from the app.
8. Click **Register Token** or **Re-register Token**.

Chapter 12: Manage Echidna configuration

Changes made through the administration console are applied immediately to Echidna, but the changes are not saved automatically. The administrator can decide when to save their changes. If Echidna is restarted, unsaved changes are lost.

The config.xml file is in the Echidna configuration folder /opt/Salt/Echidna/conf.

If the configuration has changed but hasn't been saved, the **Config** tab has an asterisk (*), as shown here;

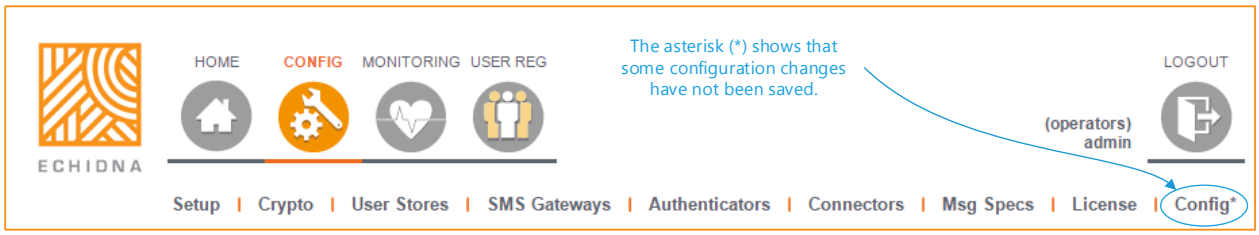


Figure 9: Screenshot of the Administration console, showing an asterisk on the Config tab

The **Config** page also shows the configuration that has changed, including the last change time.

12.1 Save configuration changes

1. Log in to the Administration console.
2. Click **CONFIG**, then click the **Config** tab on the right.
3. Click **Save Current**.

12.2 View the configuration

Echidna includes a built-in configuration viewer that allows administrators to view the entire configuration of the server in a single web page. This page displays the structural hierarchy of the XML tags as they would appear in an XML configuration file.

1. Log in to the Administration console.
2. Click **CONFIG**, then click the **Config** tab on the right.
3. Click one of these links:
   - The **current configuration** link in the sentence that begins “Export the current configuration…”.
   - The **annotated configuration here** link. This page contains links to the pages that are used to edit the corresponding elements.

12.3 Export the current configuration as a file

1. Log in to the Administration console.
2. Click **CONFIG**, then click the **Config** tab on the right.
3. Right-click the **current configuration** link in the sentence that begins “Export the current configuration...”.
4. Choose the option that saves a file, then choose a location for the file and click **Save**.
5. View **config.xml** in an XML editor or a text editor.

12.4 Import a configuration file

If the Echidna configuration has been saved as a file and then edited, the entire configuration can be imported, or only some components.

To import a configuration set for Echidna that has been saved in a file:

1. Log in to the Administration console.
2. Click **CONFIG**, then click the **Config** tab on the right.
3. Click **Choose File**.
4. Browse to the directory that contains the XML configuration file, then click **Open**.
5. Click **Import**.
6. Choose how much of the configuration to import:
   - To import only some components, check the components to add, and then click **Import Selected**.
   - To import all of the components and merge them with the existing configuration, click **Import All**.
   - To replace the entire existing configuration with the imported configuration, click **Replace All**.

If the local keystore files have changed since the original configuration was exported, or if the configuration did not originate from the same server, any encrypted values may not be properly restored. In that case, use the configuration pages to reset the passwords and shared secret values.

12.5 Backup and recovery

Echidna can import and export the entire configuration as an XML file. This allows Echidna administrators to perform routine backups of the server's configuration, which can later be restored to the server for the purpose of disaster recovery. Alternatively, backing up can be handled by taking a snapshot of the entire system.

12.5.1 Snapshot the appliance

The administrator can take a snapshot of the Echidna appliance. Although this backs up the entire system including keystores, the file size is very large.

12.5.2 Back up the Echidna configuration

If the administrator saves the configuration as a file, it can be imported later for recovery. However this does not restore the keystores.

The configuration file config.xml contains the Echidna services engine configuration, but sensitive configuration values including bind passwords and shared secret values are encrypted. The default configuration is to use JKS and JCEKS format keystore files in the same configuration folder. If the embedded Apache Derby database is being used, the default location for that is also under the configuration folder (/opt/Salt/Echidna/conf/ConfigStore).

To back up the complete state of the Echidna engine, the entire configuration folder should be archived, but to reduce the exposure of the sensitive configuration material the archive should also be encrypted. One way to achieve this is to run the following command:

```
tar cvzCf /opt/Salt/Echidna - conf | openssl aes-256-cbc -salt -k some-secret-passwd > /tmp/echidna-backup-`date --rfc-3339=date `.bin
```

This could then be restored to a recovery folder like this:

```
openssl aes-256-cbc -d -k some-secret-passwd -in /tmp/echidna-backup-yyyy-MM-dd.bin | tar xvzf -
```

When Echidna writes changes to the underlying XML or keystore files, Echidna also creates a backup of those files in a backup subfolder. This allows manual recovery if necessary.

If /opt/Salt/Echidna/conf/config.xml is to be updated, the old version is copied to a file like /opt/Salt/Echidna/conf/backup/config.20140911_231009001.xml (with the date and time in the filename) before config.xml is changed. If the utility.jks keystore file in the same folder is to be updated, the old version is copied to backup/utility.20140905_005418851.jks first.

12.5.3 Restore the Echidna configuration

1. Stop Tomcat.
2. Copy the old file(s) back to the original name.
3. Restart Tomcat.

12.6 The Setup wizard

The Installation Guide describes how to use the Setup wizard to set up the most common configuration options. When an administrator clicks on the Setup tab, Echidna detects whether any of the basic configuration elements are missing, and shows the Setup step to address that.

The administrator can visit previously configured steps to modify the configuration. To apply the choices on a Setup page to the current configuration, click **Next**.

The Setup wizard preserves the existing configuration as much as possible, and merges the changes. To view the exact changes, use the **Config** tab.

12.7 Warnings

If warnings mentioning **localhost.cred** appear, this indicates that an Echidna client is not defined for localhost. The file **localhost.cred** contains the localhost client password.

12.8 Time-based tokens stopped working

The following time-based tokens rely on the server time:

- Salt mCodeXpress
- OATH TOTP
- OCRA with time input

If some or all of these tokens stop working, the server time might have jumped. See Time synchronisation on page 16 for more information.

12.9 Licensing for Echidna

Echidna is installed with an **evaluation license**. This is valid for one month from the time the server is configured. The evaluation license limits Echidna to only two tokens for each authentication method, and no connection to an SMS gateway.

This evaluation license can be extended for organizations that need to keep testing Echidna and its capabilities. Contact the Echidna supplier to arrange for an extended evaluation period.

To remove the limitations of the evaluation license, register Echidna by applying for a permanent license.

Each **permanent license** permits a certain number of tokens to be used. Each licence can include an overall token limit, plus a limit for each authentication method. For example, if an organisation's license has an overall token limit of 100 plus limits of 100 mCodeXpress tokens and 10 OATH TOTP tokens, Echidna allows up to 100 mCodeXpress tokens and up to 10 OATH TOTP tokens, provided the total does not exceed 100.

The Salt mCodeXpress mobile app is free and has no expiry date. An organisation's users can install Salt mCodeXpress on as many mobile devices as they like. When a user registers their Salt mCodeXpress app with their organisation, Echidna counts it as a token, for licensing purposes.

12.9.1 How the license request process works

To receive a permanent license, use the Administration console to create a license request.

1. An administrator configures Echidna to include only the authenticators that will be used in the productions system.
2. The administrator uses the steps in Apply for a license on page 133 to create a license request and send it to the Echidna supplier.
3. If required, the Echidna supplier contacts the organization to discuss the license requirements.
4. The Echidna supplier works with Salt Group to generate a new Echidna license.
5. The Echidna supplier sends the new license to the administrator.
6. The administrator uses the steps in Add a license to Echidna on page 133 to update Echidna with the new license.

12.9.2 Apply for a license

Use these steps to create a request for a permanent Echidna license.

1. Log in to the Administration console.
2. Ensure that the currently configured authenticators are the ones that are required for the permanent license.
3. Click **CONFIG**, then click the **License** tab.
4. On the left, click **request new**.
5. Enter the following information:
   - **Organization Name**
   - **Validity Days:** The number of days for which the license will be valid.
   - **mCodeXpress Registered Token Limit:** The number of mCodeXpress tokens permitted.
   - **OATH Registered Token Limit:** The number of generic OATH tokens permitted. For example, if this number is 100, the organization can have 100 OATH TOTP and OATH HOTP tokens in total.
   - **Registered SMS Limit:** The number of SMS tokens permitted.
   - **Aggregate Token Limit:** The total number of tokens permitted, across all token types.
   - **Enable Brokered Authentication:** Check this box to enable brokering to a third-party authentication server.
6. Click **Next**.
7. Make any changes, then click **Update**.

   A license request is generated. The license request includes details of the authenticators that are currently configured.
8. Send the license request in one of these ways:
   - Click **Email license request** to open a pre-written email in the email client.
   - Click **Download license request** and then attach the request file to an email.
9. Include any further requests in the body of the email, then send the email to sales@saltgroup.com.au or to the Echidna supplier.

12.9.3 Add a license to Echidna

The Echidna supplier emails back a license file, which is an XML file that updates the Echidna configuration.

1. Log in to the Administration console.
2. Click **CONFIG**, then click the **Config** tab on the right.
3. Click **Choose File**, then select the new license file.
4. Click **Import**. The elements of the license appear.
5. Select at least the **HostIdentificationConfig** and **HostLicenseConfig** elements. These elements define the main Echidna license. The other elements are relevant only for setting up messaging gateways.
6. To check that the new license imported correctly, click the **License** tab and check that the license type is listed as **registered**.

12.10 Update Echidna

Echidna is supplied as a virtual appliance. Administrators can apply updates to an existing version without creating a new virtual machine, using the steps in this section. For an upgrade to a new version, create a new virtual machine.

Echidna is a Java application that contains WAR files. To upgrade Echidna, insert new versions of three WAR file, then restart Echidna.

1. Ask the Salt Group sales representative for the latest version of the following files:
   - ROOT.war
   - SelfServ.war
   - UserSupport.war
2. Save these files on the appliance:
   a. Identify the path to each of these files.
   b. Sign in to the appliance.
   c. Use the following commands to move the files to the appliance:

      ```
      wget –c path/ROOT.war
      wget –c path/SelfServ.war
      wget –c path/UserSupport.war
      ```
3. Use the following command to stop Echidna:

   ```
   service tomcat7 stop
   ```
4. Remove the old Echidna version:

   ```
   cd /var/lib/tomcat7
   rm -rf ./work/*
   rm -rf ./webapps/*
   rm -rf ./webapps2/*
   ```
5. Set the correct permissions on the new WAR files:

   ```
   cd ~
   chmod ugo+x ROOT.war SelfServ.war UserSupport.war
   ```
6. Copy the new WARs into Tomcat:

   ```
   cp ROOT.war /var/lib/tomcat7/webapps
   cp SelfServ.war UserSupport.war /var/lib/tomcat7/webapps2
   ```
   Note that SelfServ.war and UserSupport.war are in a different location from ROOT.war.
7. Start Echidna:

   ```
   service tomcat7 start
   ```

Echidna is now updated. Note the new version number in the footer of the Administration console.

Chapter 13: How processes work

In Echidna, a process is a way of deciding what to do next and how to do it, and then doing it. Each process is triggered by an event.

Echidna has the following types of process:

Type of process	Trigger event	How the process works
User authentication process	A user enters credentials	The process uses the credentials to decide which authentication method to use.
User resolution process	A user enters credentials	The process matches the credentials with an entry in a user store.
SMS sending process	An SMS authenticator needs to send information to a user	The process reformats the phone number if necessary, and then chooses the right sending method, including the right SMS gateway.
Attribute update process	The value of a particular attribute is changed	Identifies a trigger attribute, and changes the value of another attribute when the trigger attribute is updated.

The Setup wizard creates some processes. These processes are already configured to work well in most situations.

Even when an administrator creates a new process, it contains enough information to work well.

13.1 Process conditions

A process includes at least one condition. A condition is a logical statement that can be true or false, plus an action. For example, a condition could test whether a user name is exactly eight characters long, and then choose an authentication method based on the answer.

Conditions can include the following actions:

- **SetVariableStep**: Set a context variable to the result of evaluating a given expression.
- **ExpandUserConStep** (expand user context): Set a context variable to the result-set obtained by invoking a database query.
- **ResolveUserStep**: Load a user-context from the referenced user store. Only relevant for user-resolve processes.
- **AuthenticateStep**: Attempt to authenticate a submitted credential. Only relevant for authentication processes.
- **AuthResultStep**: Terminate an authentication process, setting the result to the given named result and parameters.
- **UseGatewayStep**: Attempt to send a message via a messaging gateway. Only relevant for message-sending-processes.

Not all actions are relevant to all types of process.

13.1.1 Types of process

Each condition is one of the following types:

- **Basic conditions**: These compare two values to see if they match.
- **Composite conditions**: These do not compare values, but instead they evaluate their sub-conditions.

Only basic conditions include the following:

- **Reference values**: An expression that is evaluated to a single value or a set of values. See the next chapter for details of the expression language.
- **Match values**: The value that is matched with each of the result values from the reference expression.

Both basic and composite conditions include the following

- **Sub-conditions**: Evaluated to determine whether the parent condition succeeds or not.
- **Sub-actions**: Invoked if the condition evaluation succeeded (that is, if it evaluated to true).

This table lists the operators that are used in **basic** conditions:

Operators used in basic conditions	**Description**
AND	The condition succeeds if all the sub-conditions succeed. The sub-conditions are evaluated in order and the evaluation stops as soon as one sub-condition fails.
OR	The condition succeeds if any of the sub-conditions succeed. The sub-conditions are evaluated in order and the evaluation stops as soon as one sub-condition succeeds.
NOT	The condition succeeds if (one of) the sub-conditions fail. If there is more than one sub-condition present the operation is a logical **NAND**. In this situation, the sub-conditions are evaluated in order and the evaluation stops as soon as one sub-condition fails.
ALWAYS	An alias for AND with no sub-conditions. The condition always succeeds.
CHOOSE	CHOOSE is like a case statement or an if-then-else statement. The sub-conditions are evaluated in order and the evaluation stops as soon as one sub-condition succeeds. The sub-conditions are expected to have their own sub-actions. If none of the sub-conditions succeed then the sub-actions of the CHOOSE condition are invoked (the otherwise clause). Logically this is equivalent to a **NOR** condition, except there is a special treatment for authentication-processes: When one of the sub-conditions succeeded but then had a sub-action invoking an authenticator that gave a **service unavailable** or **token-unavailable** result from that authenticator, that sub-condition is treated as **failed** so that processing can continue with the sub-conditions to find an alternative authentication method.

This table lists the operators that are used in **composite** conditions:

Operators used in composite conditions	Description
PRESENT	The condition succeeds if the reference expression evaluates to at least one non-empty value. There is no match value.
NOTPRESENT	The condition succeeds if the reference expression evaluates to an empty set or value. There is no match value.
EXACTMATCH	The condition succeeds if at least one of the values that the reference expression evaluates to exactly matches the match value.
CASEIGNOREMATCH	The condition succeeds if at least one of the values that the reference expression evaluates to matches the match value using a case-insensitive comparison.
REGEX	The condition succeeds if at least one of the values that the reference expression evaluates to matches the regular expression specified by the match value.
GT (Greater Than)	The condition succeeds if at least one of the values that the reference expression evaluates can be parsed as an 8-byte value that is greater than the match value.
LT (Less Than)	The condition succeeds if at least one of the values that the reference expression evaluates can be parsed as an 8-byte value that is less than the match value.
GTE (Greater Than or Equal)	The condition succeeds if at least one of the values that the reference expression evaluates can be parsed as an 8-byte value that is greater than or equal to the match value.
LTE (Less Than or Equal)	The condition succeeds if at least one of the values that the reference expression evaluates can be parsed as an 8-byte value that is less than or equal to the match value.
AVAILABLE	The match value is the name of an authentication method or a defined role. The condition succeeds if there is a current user context which is a member of the role or for which the named authentication method is available for validation of credentials (OTPs). What it means to be *available* is up to the configuration and logic of each authentication method. There is no reference expression (the reference is implicitly the current user context).

13.2 Process actions

The **set context variable** action (SetVariableStep) sets a context variable to the result of evaluating a given expression. It has the following parameters:

Parameter	Description
Var Name	The name of the context variable to set.
Expression	The context expression to evaluate to obtain the variable value.
Delimiter	Flattens a collection of items into a single text value before assigning to the variable. Leave this empty if the variable gets the collection as it is.
Append	If set to true, append the new value to the list of any existing values.
Sensitive Value Handling	Determine how to treat values that appear to be sensitive when setting the variable. The default value is KEEP, which means the value is preserved. Other options are MASK and REMOVE means sensitive values are replaced with *** or removed altogether – which may be relevant if the expression is being created for logging purposes.
Setters	A further list of variable names and expression values to set.

The **expand user context** action (ExpandUserConStep) sets a context variable to the result-set obtained by invoking a database query. It has the following parameters:

Parameter	Description
Context Variable Name	The name of the context variable to set with the results of the database query.
Persistence Configuration	The persistence configuration, out of which the first named query is the one invoked to obtain the result set. See How persistence works on page 142.

The **resolve user** action (ResolveUserStep) has only one parameter:

Parameter	Description
Store name	The user store, to try to resolve a user context from the given username

The **authenticate user** action (AuthenticateStep) has the following parameters:

Parameter	Description
Authenticator Name	The name of the authenticator to invoke if this process step is triggered.
Password Combo	The method of combining user-store password with OTP in the target authenticator (if it is OTP-based), overriding any default value specified in the authenticator. Sometimes the same authenticator (e.g. Salt mCodeXpress) may be used from two different access points, one of which requires combination with user store passwords, and one which does not. This option allows the same authenticator to be used for both.

The **use gateway** action (UseGatewayStep) has the following parameters:

Parameter	Description
Mobile Attr	The expression (usually containing a reference name of a request context variable) that when evaluated is expected to provide the message recipient identifier (the mobile phone number for SMS, the email address for email). The default is ${mobile}.
Gateway Name	The name of the configured messaging (SMS) gateway to invoke.

The **authentication result** action (AuthResultStep) has the following parameters:

Parameter	Description
Authentication Result Name	The named authentication result that is generated if this process step is triggered.
Result Params	The parameter settings for the authentication result that is generated if this process step is triggered. • **Key**: The name of a format parameter. • **Value**: An expression for the value. To see the available parameters for a given Authentication Result Name (e.g. NO_AUTHMECH_SELECTED), look in the **Result Specifications** section of the Config > Msg Specs page for the named result. That table identifies the log-message-spec and the reply-message-spec which can be clicked to navigate to the definition of those formats and the available format parameters. The most common parameters (USERID, USERNAME, USERDOMAIN and USERSTORE) are automatically provided when the result is constructed.
Params	The list of generic key-value configuration pairs.

13.3 Authentication processes

In addition to the common process conditions and actions described above, authentication processes also define roles and lockout parameters, and allow identification of a preferred authentication mechanism user attribute:

Parameter	Description
User Selected Preferred Mechanism Attribute	This group of properties identifies an attribute in the user store that is intended to let the end user or the support staff control which authentication method should be the current default for a user. If this is defined, the User Support console and the Self Service console let the attribute be set to the **external identifier** name of any of the configured authentication methods that are permitted to the user. The rest of the authentication process is not constrained to honour this attribute, but the normal pattern is to skip any authentication methods that don't match the current value in this attribute.
User Roles	This is the section where the default application roles and the organisation's custom roles can be defined.
Lockout Configuration	The lockout configuration augments the normal invocation of the process steps: • Before the normal steps are invoked, a check is made to see if the attempt limit has been exceeded for the current resolved user. If is has been exceeded, the LOCKEDOUT_USER result is immediately set and the rest of the process is not invoked. If it has not been exceeded, the **failureCount** request context variable is set to the current value and the normal process is invoked. • After the normal steps are invoked, a check is made to see if the result matches the lockout criteria. If it matches, the lockout-counter is incremented for that user ID.

Chapter 14: How persistence works

Several areas of Echidna functionality may use a database to persist (store and retrieve) information to database tables, including the following:

- Token data records for Salt mCodeXpress or OATH tokens
- User and group records for JDBC-backed user stores
- Audit event records generated by incoming service requests and outgoing gateway invocations

Common to all these usages is the need to be somewhat flexible about mapping data to tables and columns (particularly for user-data tables shared with existing applications), and to protect the confidentiality and integrity of the stored data.

To support this, any configuration elements that may leverage database persistence do so via a common set of *persistence configuration* elements, which are described here.

14.1 Datastore (JDBC) connections

While the rest of the persistence configuration controls what goes into the database, the Datastore Connection Name controls which database it goes in. It is a reference to one of the configurations in the JDBC Connections list, the default being LocalStoreUnit.

Each of the JDBC connections are configured as follows:

Parameter	Descriptions
Connection Name	An identifier for the database connection to allow it to be referenced in persistence configurations.
Default Schema Name	When creating new table definitions in persistence configurations using this connection, this schema name is used as the default schema name for the table.
Bind Credential	The username and protected password that the JDBC driver may need when making a new connection to the database. For Microsoft SQL server using integrated Windows authentication, this is not required.
JDBC Driver Class	The name of the JDBC driver class, which ensures that the relevant JDBC support JAR files are loaded and the relevant JDBC driver is registered with the JDBC DriverManager.
JDBC Connection URL	The specification for connecting to the database, which typically includes a server name, port, and database name. For the exact format, refer to the JDBC driver documentation for the database in use. See the common patterns table, below.
DBCP Connection Pooling	Enabling this section allows the Apache Commons Database Connection Pooling library to manage the database connections in a connection pool. It is recommended to use this option unless the JDBC driver already performs connection pooling.

Some common patterns for selected database types are:

Database	Driver Class	Sample JDBC URL
Apache Derby (Embedded)	`org.apache.derby.jdbc.EmbeddedDriver`	`jdbc:derby:directory:ConfigStore;create=true`
Oracle	`oracle.jdbc.driver.OracleDriver`	`jdbc:oracle:thin:@//localhost:1521/EchidnaStore`
Microsoft SQL Server	`com.microsoft.sqlserver.jdbc.SQLServerDriver`	`jdbc:sqlserver://localhost:1433;databaseName=EchidnaStore`
MySQL	`com.mysql.jdbc.Driver`	`jdbc:mysql://localhost:3306/EchidnaStore`

14.2 Protecting data confidentiality

When defining the persistence tables, the configuration for each column may specify that values in that column should be protected. In that case, the nominated Data Confidentiality Protector is used to encrypt values before writing to the database and decrypt the values when they are read back out.

14.3 Protecting data integrity

When defining the persistence tables, one column may be nominated as a TEMAC (Tamper-Evident Message Authentication Code) column.

If this is done, a MAC of the rest of the row data is made when appending or updating the table, using the cryptographic key referenced by the Data Integrity Protector name. When a query returns a row from the database, the TEMAC value is recalculated and compared with the stored value to detect any changes that may have occurred through some mechanism other than Echidna. If the values don't match, the row is considered *tampered with* and the values are not trusted.

14.4 Tables, updates and queries

The rest of the persistence configuration defines the SQL query and update statements that Echidna uses to interact with the database, and the table definitions for the tables referenced in the queries or updates. If the JDBC connection credentials have the appropriate permissions, Echidna can also be used to create or update the tables in the database to reflect the configured definitions.

Chapter 15: Expression language

Many configuration elements (such as the reference attribute of a process condition, the body of an HTTP service call, or the text of a persistence query or update) allow the value to include references to expressions that are evaluated each time the value is employed. References to expressions are surrounded by **${** and **}**.

The expressions may refer to attributes from the user store, from the service-request or result or to request context variables set in previous **set-variable** process steps. Expressions are potentially multi-valued, meaning that they can evaluate to a list of values rather than just a single value. For example, the AD user store attribute **memberOf** lists the names of all the groups a user has membership of.

Supported expressions are described in the following table:

Expression	Description
request.User-Name request.attname	For RADIUS authentication requests, **attname** is the name of the RADIUS request attribute, the most common being **User-Name**. For web service authentication calls, the parameters are mapped to the same attribute names.
client.name	The name of the Echidna client that initiated the request.
userContext.attname	A user attribute from the resolved user context.
userContext.inRole[role-name]	If the user context has membership of the identified role, this evaluates to the full name of the role, otherwise it evaluates to **empty**.
result.spec.attname	If an authentication result has been generated in the current context, this evaluates to the details about the particular result type: • **name**: The result name like SUCCESS or CRED_VALIDATION_FAILED. • **type**: The category – ACCEPT, REJECT, CHALLENGE or DISCARD. • **userMessage**: The reply-message for display to the end user. • **logMessage**: The log message for recording in the event log.

Expression	Description
result.attname	If an authentication result has been generated in the current context, this evaluates to any of the result-specific attributes of the result. The common attributes set on most results are: • **AUTH_NAME**: The name of the authenticator being used to process the request. • **PACKETID**: The request packet identifier (a small numeric value, 0 – 255 for RADIUS, 0 – 65536 for web services). • **USERDOMAIN**: The domain specified by the user or of the resolved user store. • **USERSTORE**: The name of the resolved user store. • **USERID**: The userid specified by the user, or of the resolved user context. • **USERNAME**: The full name of the resolved user store entry. For AD or LDAP, userid is similar to **tester1** while username is similar to **CN=Tester One, OU=Testers, O=Control**.
varname	The value of any context variable previously set with a process **set-variable** step. In addition, there are a few special dynamic variables that are generated on demand as follows: • **HTTP_Date**: The current time in a format that can be used for HTTP date headers (EEE, d MMM yyyy HH:mm:ss 'GMT'). • **TS_Date**: The current time in a short timestamp format (yyyyMMdd_HHmmss_SSS). • **uintseq**: The next positive integer value from an internal but transient sequence counter value that can be used to generate unique reference numbers if they are needed. In addition, if there is currently a defined **bindCred** context variable (which is the case when an outgoing HTTP service call is being made), the following are also defined: • **bindUser**: The userid part of the bind credentials. • **bindPassword**: The plaintext password part of the bind credentials. • **HTTP_BasicCred**: The bind credentials formatted into the HTTP Basic-Authentication format, which is the word **Basic** followed by the base64 encoding of **username:password**.

Chapter 16: Web services API

Echidna has a web services API. Client applications can use this API to communicate with Echidna. This section describes how to access information about the API.

16.1 Ensure that Echidna has an HTTP access point (if required)

Usually this is set up by default during installation. However, if the installer deselected that option, it is not set up. If required, set up an HTTP access point:

1. Follow the instructions in Add an access point for client requests on page 107.
2. Use the following values:
   - **New access point bind host**: 0.0.0.0
   - **New access point port** : 1888
   - **Select access point type**: TCP (HTTP) Authentication Services
   - **Name of service to access**: webServices
3. Ensure that the access point is started.

16.2 Create a client

1. Log in to the Administration console.
2. Click the **Connectors** tab.
3. In the **Clients** section on the left, click **new**.
4. In the **New RADIUS Client Name or IP Address** box, enter a username.
5. Click **Next**.
6. Create a new password by entering it into the **New Password** and **Confirm Password** boxes.
7. Click **Update**.

16.3 Log in to the API

1. In a web browser, go to the address for the API.

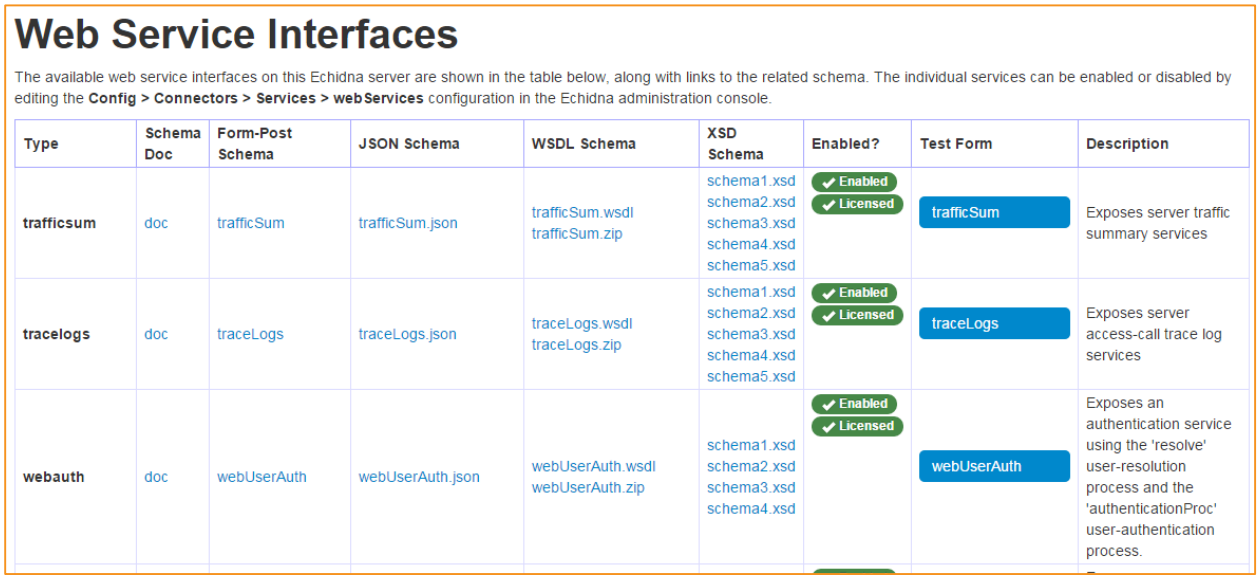
   Use the address of the Echidna server and the port number of the HTTP web services.

   For example, if the Administration console is at **https://198.51.100.62:8443** and the HTTP access point port is **1888**, use the following address to access the web services API:

   http://198.51.100.62:1888

2. Log in using the username and password that were created for the new client.

   The Web Service Interfaces page appears:



Web Service Interfaces

The available web service interfaces on this Echidna server are shown in the table below, along with links to the related schema. The individual services can be enabled or disabled by editing the **Config > Connectors > Services > webServices** configuration in the Echidna administration console.

Type	Schema Doc	Form-Post Schema	JSON Schema	WSDL Schema	XSD Schema	Enabled?	Test Form	Description
trafficsum	doc	trafficSum	trafficSum.json	trafficSum.wsdl trafficSum.zip	schema1.xsd schema2.xsd schema3.xsd schema4.xsd schema5.xsd	Enabled Licensed	trafficSum	Exposes server traffic summary services
tracelogs	doc	traceLogs	traceLogs.json	traceLogs.wsdl traceLogs.zip	schema1.xsd schema2.xsd schema3.xsd schema4.xsd schema5.xsd	Enabled Licensed	traceLogs	Exposes server access-call trace log services
webauth	doc	webUserAuth	webUserAuth.json	webUserAuth.wsdl webUserAuth.zip	schema1.xsd schema2.xsd schema3.xsd schema4.xsd	Enabled Licensed	webUserAuth	Exposes an authentication service using the 'resolve' user-resolution process and the 'authenticationProc' user-authentication process.