

LinOTP Documentation

Release 2.5.2

LSE Leading Security Experts GmbH

August 02, 2013

Contents

I LinOTP Management Guide **3**

1 Introduction **5**
1.1 System Overview . . . 5
1.2 Components . . . 5
1.3 LinOTP core . . . 5
1.4 OTP Calculation . . . 5
1.5 UserIdResolver . . . 7
1.6 Authentication Modules . . . 7
1.7 Management Clients . . . 7
1.8 Features of different management clients . . . 8
1.9 Licenses . . . 8

2 Management Web Client **9**

3 Configure Default settings, Resolvers and Realms **11**
3.1 Default Settings . . . 11
3.2 Configuring UserIdResolvers . . . 12
3.3 Configuring Realms . . . 14

4 System config **15**

5 Supported tokens **17**
5.1 Recommended Mobile Apps . . . 18

6 Managing Tokens **21**
6.1 Import tokens . . . 21
6.2 Viewing users in certain realms . . . 26
6.3 Viewing tokens . . . 27
6.4 Assign tokens . . . 27
6.5 Set OTP PIN . . . 30
6.6 Enrolling tokens . . . 30
6.7 Manage tokens . . . 40
6.8 Set token realm . . . 41
6.9 Token info . . . 41
6.10 Lost token . . . 43
6.11 Get serial by otp . . . 44
6.12 Get otp . . . 45

7 Policies **47**
7.1 Selfservice policies . . . 47
7.2 Admin Policies . . . 50
7.3 System policies . . . 51
7.4 License policies . . . 52
7.5 Enrollment Policies . . . 52
7.6 Authorization Policies . . . 55

7.7 Authentication Policies . . . 57
7.8 Gettoken Policies . . . 59
7.9 Audit Policies . . . 59
7.10 OCRA Policies . . . 59
7.11 Policy checker . . . 60
7.12 Importing and exporting policies . . . 61
7.13 Users in policies . . . 61
7.14 Clients in policies . . . 62
7.15 Best practice - policy example . . . 63

8 Audit Trail **65**
8.1 Logging Scopes . . . 65
8.2 Viewing the Audit Trail . . . 67
8.3 Configuration . . . 67
8.4 Searching the Audit Trail . . . 68

9 SMSProvider for SMS OTP Tokens / Mobile TANs **69**
9.1 Overview . . . 69
9.2 SMSProvider . . . 69
9.3 SMSProviderConfig . . . 70
9.4 SMSProviderTimeout . . . 72

10 Security Module **73**
10.1 PKCS11 module and SafeNet LunaSA . . . 73
10.2 Password handling . . . 73

11 LinOTP as OpenID Provider **75**
11.1 Comfortable identity links . . . 75

12 Retrieving OTP values **77**

13 Self service portal **79**
13.1 Self service portal Imprint . . . 79

14 Tools **81**
14.1 linotp-sql-janitor . . . 81
14.2 linotp-tokens-used . . . 81
14.3 linotp-backup . . . 81
14.4 linotp-restore . . . 81
14.5 linotp-convert-token.pl . . . 81
14.6 linotp-decrypt-otpkey . . . 82
14.7 LinotpLDAPProxy.pm . . . 82

15 Backup and Restore **83**
15.1 Disaster Recovery . . . 83

16 PCI DSS **85**

17 Background information and concepts **87**
17.1 Users with no token . . . 87
17.2 UserIdResolvers and Realms . . . 87

18 Usage scenarios **91**
18.1 Protection levels . . . 91

19 Troubleshooting **93**
19.1 LDAP/Active Directory Connection . . . 93
19.2 Testing the Freeradius Server . . . 93
19.3 RADIUS Authentication fails . . . 93
19.4 RADIUS server log file . . . 93

19.5 Access to Web API . . . 94
19.6 OTP Authentication . . . 94
19.7 LinOTP server log file . . . 94

II LinOTP Installation Guide 97

1 Supported Operating Systems 99
1.1 Supported Database systems . . . 99

2 Checklist 101
2.1 Decide which distribution you want to use . . . 101
2.2 Decide which token database you want to use . . . 101
2.3 SSL certificate . . . 101
2.4 LinOTP Admin . . . 101
2.5 User databases . . . 102
2.6 User tokens . . . 102
2.7 Authentication . . . 102
2.8 Location in your network . . . 102

3 Server installation 105
3.1 LinOTP Virtual Appliance installation . . . 105
3.2 Installing Debian packages . . . 107
3.3 LinOTP Server Installation – the tar.gz, virtualenv and pip way . . . 109
3.4 LinOTP on Univention Corporate Server UCS 3.1 . . . 114
3.5 Configuration background information . . . 117

4 Installing Management Clients 125
4.1 Install the Debian packages . . . 125
4.2 Install python packages . . . 125
4.3 Management Client Installation on Windows . . . 126

5 Installing Authentication Modules 129
5.1 Enabling PAM authentication with pam_linotp . . . 129
5.2 Enabling RADIUS authentication with rlm_linotp2 . . . 130
5.3 Enabling RADIUS authentication with Radiator . . . 134
5.4 Enabling RADIUS authentication with rlm_perl . . . 134
5.5 Enable SAML Identify Provider . . . 135

6 Customization 137
6.1 Templates . . . 137
6.2 Style Sheets . . . 137
6.3 Changing the Logo . . . 137

7 Database connection 139
7.1 Running LinOTP with DB2 on RedHat . . . 139
7.2 Running LinOTP with Oracle on RedHat . . . 141
7.3 Running LinOTP with Microsoft SQL . . . 142
7.4 Special parameters for database connections . . . 144

8 Security Modules 147
8.1 Defining Security Modules . . . 147
8.2 Defining SafeNet LunaSA . . . 147
8.3 Setting up SafeNet LunaSA . . . 148
8.4 Create AES Keys . . . 153
8.5 Backup and restore with LunaSA . . . 153
8.6 Setting up HA and Load balancing for LunaSA . . . 154
8.7 Managing Passwords with LunaSA . . . 156

9 Integration examples **159**
9.1 OTP Authentication with Apache2 . . . 159
9.2 Firewall integration . . . 160
9.3 Authentication with third party OTP solutions . . . 160

10 Updates **167**
10.1 Updating from LinOTP 2.5.1 to LinOTP 2.5.2 . . . 167
10.2 Updating from LinOTP 2.5.0 to LinOTP 2.5.1 . . . 169
10.3 Updating from LinOTP 2.4.4 to LinOTP 2.5.0 . . . 170

11 Migrating from LinOTP 1.3 or LinOTP 1.0 **173**

12 Security advisories **175**
12.1 Locked Users . . . 175
12.2 UserIdResolver . . . 175

13 Troubleshooting **177**
13.1 Connecting to userstores . . . 177
13.2 Distribution not Found: LinOTP2 . . . 177
13.3 SQLAlchemy error . . . 177
13.4 Access denied for user root@localhost . . . 177

III LinOTP User Guide **179**

1 Introduction **181**

2 Workflows **183**
2.1 Using mOTP token . . . 183
2.2 Using eToken Pass . . . 187
2.3 Enrolling OATH Token or Google Authenticator . . . 191

IV LinOTP Appliance Manual **193**

1 Getting started **197**

2 Configuring network settings **199**

3 Managing LinOTP token administrators **201**

4 LinOTP debug logging **203**

5 Configuring the RADIUS access to the LinOTP appliance **205**
5.1 Detailed settings . . . 205

6 Working with configuration sets **207**

7 Root user and appadmin user **209**

8 Change the server SSL certificate **211**
8.1 New self-signed certificate . . . 211
8.2 Create Certificate Request – Externally signed Certificate . . . 211

9 Advanced settings **213**

10 Redundant setup **215**
10.1 Setup redundancy . . . 215
10.2 Known Errors . . . 216
10.3 Reverting Redundancy . . . 216

11 The support file 217

12 Updates 219

13 Backup and restore 221

14 Network integration 223

15 Emergency recovery 225

V LinOTP Module Development Guide 227

1 Authentication interfaces 231
1.1 Validate Controller . . . 231
1.2 OCRA Controller . . . 233
1.3 Example for authentication integration . . . 236

2 Administrative Interfaces 241
2.1 Admin Interface . . . 241
2.2 Admin Controller . . . 242
2.3 System Controller . . . 248
2.4 License Controller . . . 252
2.5 Session protection . . . 253

3 Tokens 255
3.1 The GUI file . . . 255
3.2 The token file . . . 258

4 UserIdResolver Modules 261

5 Licenses 263
5.1 AGPLv3 . . . 263

VI Indices and tables 273

Index 277

Preface

By using LSE LinOTP you decided to use a flexible, modern authentication system.

Congratulations on your choice!

LSE LinOTP is a solution for authenticating with one time passwords. The runtime components of LSE LinOTP are available as source code, which gives you the possibility, to make your own adaption or to conduct code reviews on the software. LSE LinOTP is originally based on GNU/Linux but as it is based on Python, it might also run on other operating systems.

LSE LinOTP is lean and very gentle on resources. It is scalable from small installations up to world-spanning, redundant enterprise installations. A special Community Edition is also available at http://www.linotp.org.

This manual is divided into 5 main parts.

1. The *LinOTP Management Guide* (page 5) describes the managing of the users, the realms and the tokens.
2. The *LinOTP Installation Guide* (page 99) explains how to install the LinOTP Application in different, individual ways.
3. The *LinOTP User Guide* (page 181) shows the user how to use the self service portal.
4. The *LinOTP Appliance Manual* (page 195) deals with the LinOTP Appliance. No matter if you are using the hardware appliance or the virtual appliance, here you can find how to setup and configure the appliance.
5. If you plan to adapt or extend LinOTP you can develop your own modules. How to do this is described in *LinOTP Module Development Guide* (page 229).

Getting Support

With licensing the LSE LinOTP 2 Enterprise Edition from LSE Leading Security Experts GmbH you are entitled to get support.

Support via e-Mail: support@lsexperts.de (support@lsexperts.de)

Telephone: +49 6151 86086 - 115

For further information, special tips and tricks and howtos you may take a look at http://www.lsexperts.de/kundenportal.html.

I LinOTP Management Guide

1 Introduction

1.1 System Overview

LSE LinOTP is a framework that provides most flexible authentication with One Time Passwords (OTP). The dimension of flexibility is shown in this section.

1.2 Components

LSE LinOTP consists of several component types.

The components are shown in *Components of LinOTP* (page 6).

The components are loadable modules which can be used depending on the setup with the LinOTP core. Since the interfaces for component types are well defined, it is straightforward to implement new components without any impact on the LinOTP core. Such new components can be easily loaded during runtime.

1.3 LinOTP core

This is the central server part, the LinOTP core. LinOTP is implemented in Python and well tested with Python 2.7 and will also run with Python 2.6. It uses Pylons for the communication of the other components with the core. Thus the other components like management clients and authentication modules will issue HTTP requests to communicate with the LinOTP core. LinOTP stores all token information in an SQL database. MySQL, PostgreSQL, SQLite, Oracle and DB2 were tested successfully.

1.4 OTP Calculation

The LinOTP core is capable of using different OTP algorithms for calculating the OTP values. Each OTP token is stored with its token type, that identifies how the OTP value is calculated.

At the moment the following types are supported:

- event based HOTP [1],
- time based TOTP [2],
- time based mOTP [3],
- LSE simple pass token,
- SMS OTP,

[1] RFC 4226
[2] RFC 6238
[3] http://motp.sourceforge.net

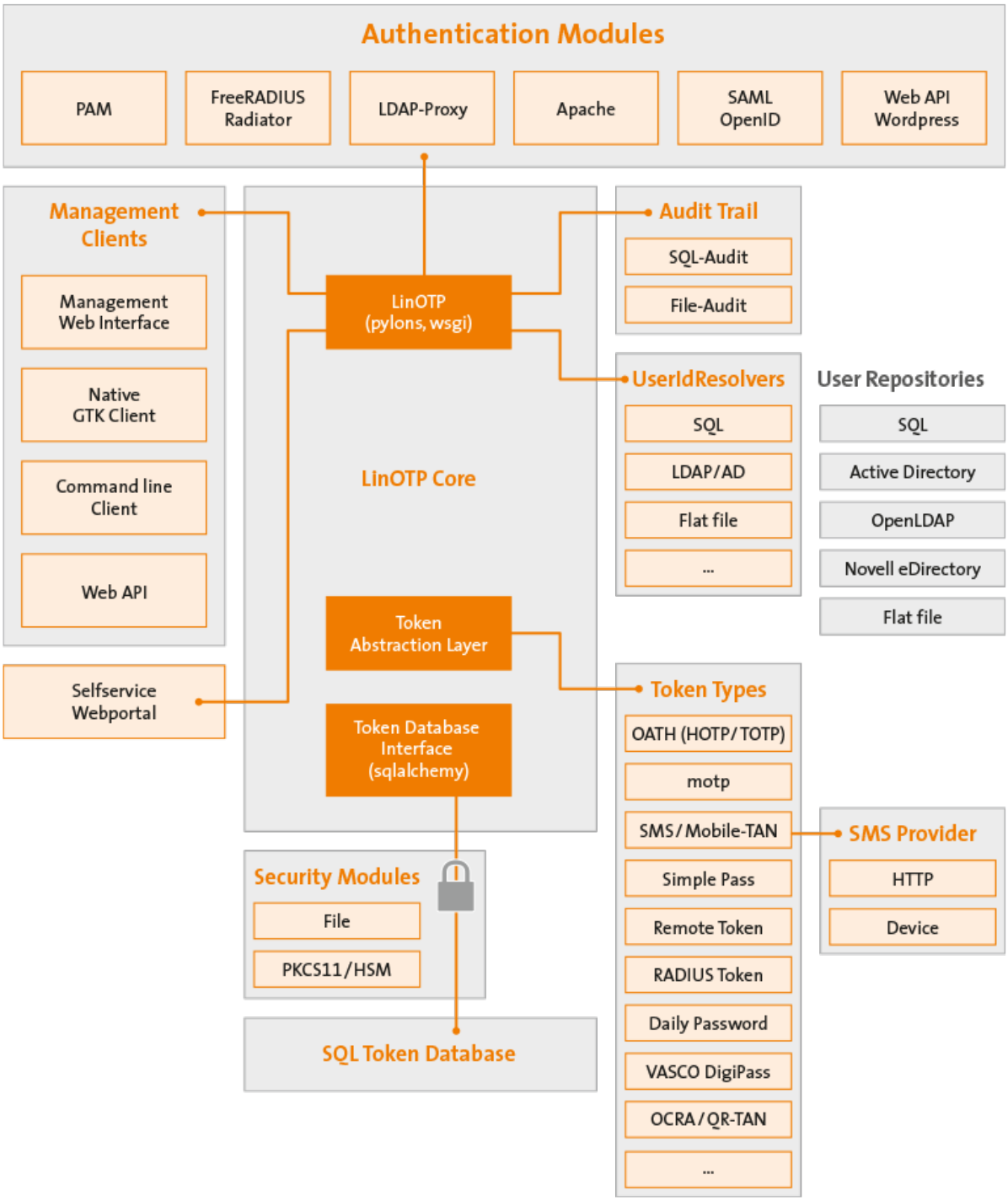


Figure 1.1: *Components of LinOTP*

- RADIUS Token,
- Remote Token,
- Daily Passwords,
- Vasco Digipass Tokens.

New modules for new tokens can be plugged in easily. LinOTP is OATH certified for HOTP and TOTP.

1.5 UserIdResolver

LinOTP uses external userstores to identify users to whom tokens get assigned. LinOTP does not modify the userstore. A UserIdResolver has a well defined interface. So a new useridresolver can be plugged into LinOTP so that users from another userstore can be used. LinOTP can use several userirdresolvers at the same time. This LinOTP Enterprise Edition provides a Passwd UserIdResolver to access users from flat files like /etc/passwd,an LDAPIdResolver to use users from LDAP directories like OpenLDAP, Active Directory or Novell eDirectory and SQLUserIdResolver to access users in SQL databases.

1.6 Authentication Modules

LinOTP does not bind you to any authentication method. Although RADIUS (Remote Authentication Dial In User Service) is an often used protocol, it might not fit all the needs or might be sometimes to costly to set up. So LinOTP also provides an interface for authenticating users. At the moment LinOTP Enterprise Edition provides an authentication module for FreeRADIUS and for the Unix PAM stack (Pluggable Authentication Module). Additionally LinOTP also provides a simple web API.

Again as the LinOTP authentication interface is very lean, other authentication modules can be implemented easily.

1.7 Management Clients

The LinOTP server can be managed in several different ways. There is a command line client for Windows and Linux, a native GUI for Windows and Linux (see Figure 2) and starting with LinOTP server 2.2 a Web UI (see Figure 3). All clients can be used for all administrative and token management tasks. Using this management interfaces the LinOTP server and its UserIdResolvers can be configured. Tokens can be imported, enrolled, assigned or disabled. For a detailed feature comparison of the different management clients see Table 1. The management client also provides a view to the available users in the configured userstore. Of course – as LinOTP only has read access to the userstore – the users will not be managed within LinOTP.

1.8 Features of different management clients

Feature	CLI management client (linotpadm.py)	Web UI	GUI management client (glinotpadm.py)
Manage UserIdResolver	ok	ok	ok
Manage Realms	ok	ok	ok
Manage license	–	ok	ok
Enroll eTokenNG OTP	–	–	ok
Enroll mOTP Token	ok	ok	ok
Import Token XML file	ok	ok	ok
Import PSKC file	–	ok	–
Enroll/Assign eTokenPASS and other HOTP token	ok	ok	ok
Enroll/Assign TOTP tokens	ok	ok	ok
Enroll Simple PASS token	–	ok	ok
Enroll SMS token	–	ok	ok
Enroll YubiKey	ok	–	ok
Enroll Remote Token	–	ok	ok
Enroll RADIUS Token	–	ok	ok
Enroll Vasco Token	–	ok	ok
Manage Tokens (enable, disable, delete, assign, unassign)	ok	ok	ok
Reset Failcounter	ok	ok	ok
Manage Token detailed token settings	–	ok	ok
mass enroll eToken NG	ok	–	–
mass enroll Yubikey	ok	–	–

Features of the different management clients

1.9 Licenses

The LinOTP Server, the UserIdResolver (PasswdIdResolver), linotpadm.py and the authentication module pam_linotp are licensed under GNU Affero General Public License (AGPL) Version 3 (http://www.gnu.org/licenses/agpl-3.0.html). rlm_linotp2 is licensed under GNU General Public License (GPL) Version 2 (http://www.gnu.org/licenses/gpl-2.0.html). All other UserIdResolvers (LDAPIdResolver, SQLIdResolver) and the Management Client glinotpadm.py, the documentation and scripts are to be licensed from LSE Leading Security Experts GmbH.

2 | Management Web Client

Starting with LinOTP version 2.2 a management Web UI is available. It can be accessed by calling

https://linotpserver/manage

The Web UI uses javascript, JSON and the jQuery framework. It is tested with

- Firefox 3 and 11+
- Internet Explorer 7, 8 and 9,
- and Google Chrome 18.

All tasks can be done via the web client – except enrolling eToken NG-OTP. So you can decide if you want to manage your tokens either using the web client or the native management client. If you want to use the web client you can directly go on reading with setting up UserIdResolvers and setting up realms (see section *Configuring UserIdResolvers* (page 12)).

3 Configure Default settings, Resolvers and Realms

You can configure the default settings, resolvers and realms either using the Management Web UI on

https://linotpserver/manage

or start the Management Client:

```
glinotpadm.py
```

You need to configure the management client so that the client knows were to reach the server. For configuring the server connection go to the menu *Config* → *LinOTP Client Configuration*.

There you need to enter the IP address assigned to your LinOTP Server. You may choose the protocol (HTTP or HTTPS), the server name and the port. After clicking OK the client will try to connect with this new settings and retrieve the token list and the user list. This configuration is stored in the users home directory in the file *glinotpadm.cfg*.

On the tab enrollment settings you can configure the behavior when enrolling SafeNet eToken NG-OTP.

The **default Token** name will be set, when an eToken NG-OTP is enrolled without assigning the token to a user. If the token is assigned during enrollment, the token name will be set to the loginname of the user. The OTP display duration specifies, how long the OTP value will be displayed on the eToken NG-OTP, when the token button is pressed.

On the tab enrollment settings you can configure the behavior when enrolling SafeNet eToken NG-OTP. The default Token name will be set, when an eToken NG-OTP is enrolled without assigning the token to a user. If the token is assigned during enrollment, the token name will be set to the loginname of the user.

The **OTP display duration** specifies, how long the OTP value will be displayed on the eToken NG-OTP, when the token button is pressed.

You may set a **random user PIN** and a **random SO PIN** [1] during enrollment. These PINs are stored in the database in an encrypted way for later use.

Another authentication method is using client certificates. The LinOTP server then needs to be reconfigured.

3.1 Default Settings

The settings of the server are stored in the token database, which is defined in the file linotp.ini. For configuring the server settings go to the menu *Config* → *LinOTP Server Configuration*. In this configuration dialog you can configure token default values, which are set to tokens, that are enrolled or imported anew.

[1] Security Officer PIN. This PIN can be used to unlock the smartcard part of the token.

The `Sync Window` is the window size the HMAC OTP algorithm should look into when resynchronizing HMAC OTP bases tokens. This is to avoid, that LinOTP will compute forever when resynchronizing tokens with wrong OTP values.

The `Count Window` defines the value, how many additional OTP values LinOTP will compute to verify the OTP value entered by the user.

The `OTP Length` is the length of the generated OTP values. The eToken NG-OTP and also the Safeword Alpine and eToken PASS all use 6 digits. Nevertheless RFC 4226 also allows to use 8 digits, which might be used by other tokens. If you are using tokens from SafeNet, leave this as 6 digits.

If the user fails to authenticate for more than Maximum Fail Count times, the tokens of the user get locked. Locked tokens need to be removed or unlocked.

3.2 Configuring UserIdResolvers

The idea of UserIdResolvers and Realms is described in section A.16.2 on page 75. You should read this to get an idea of how to setup LinOTP in the right way.

Open *Config* → *LinOTP Server Configuration*. Within the tab "Userstores" all the UserIdResolvers that hold the user data can be defined. LinOTP only needs read access to those user stores. You can define as many UserIdResolvers as you like to. There are three possible types at the moment:

`LDAP` Which can be used to read users from LDAP servers like OpenLDAP, Active Directory and Novell eDirectory.

`SQL` Which can be used to read users from a vast number of different SQL servers.

`Flatfile` Which can be used to read the users from any flat file located on the LinOTP server. This would usually be `/etc/passwd` but can be any other file, too.

3.2.1 Configuring LDAP UserIdResolver

The LDAPIdResolver (Figure 9) can be used to resolve user loginnames via OpenLDAP, Active Directory or Novell eDirectory.

To get **redundancy**, you can configure more than one LDAP server. To do this, you may enter more than one server in the LDAP Server configuration URI. Lets say, you have got two domain controllers with the addresses 192.168.1.31 and 192.168.1.32. (You may also use FQDN [2]). Then you need to enter the line

ldap://192.168,1.31, ldap:192.168.1.32

If LinOTP fails to contact the first LDAP server, it will try to contact the second one. If the second one can be contacted, it will be set first then.

Note: Before saving your configuration, you should push the button "Test LDAP connection" to see, if your settings enable the LinOTP server to connect to the LDAP successfully. If you have got problems setting up your LDAP connection, you may monitor your LinOTP log file `/var/log/linotp/linotp.log` on your LinOTP server. This may give you some clues about problems with the LDAP connection. Also see section *LDAP/Active Directory Connection* (page 93).

`Loginname attribute` is the attribute that holds the name, the user will use to login.

`Search filter` is used to filter the user objects found within the configured base DN [3]. You can only assign tokens to those users found by the search filter.

[2] Fully qualified domain name
[3] Distinguished Name

`User Filter` in turn will be used to resolve the users the other way round. E.g. if the user logs in, he will provide the loginname. The login name will be replaced with the "%s" in the user filter, to resolve the login name to the distinguished name of the user object in the LDAP.

`Attribute mapping` will fetch more user information from your LDAP server like:

username, phone, groups, mobile, email, surname, givenname

The required attribute is username. All other attributes are optional.

Using the button "Preset Active Directory" you can change the default LDAP attribute definition to fit the needs in an Active Directory environment. In this case you will not need to change the attribute definition manually. After having entered all necessary values, you need to push "Apply" to store the changes. The changes will be sent to the server as soon as you push the button "OK" of the Configuration Dialog.

Note: When configuring a LDAPIdResolver for connecting to Active Directory and your BaseDN is set to your top level domain components like *dc=mydomain,dc=com* you should also select the option "No anonymous referral chasing".

3.2.2 Configuring SQL UserIdResolver

The SQL UserIdResolver can be used to resolve loginnames via SQL databases. You need to enter a valid database driver like sqlite, mysql, postgresql, oracle, sapdb, mssql as described in the sqlalchemy documentation:

http://www.sqlalchemy.org/docs/dbengine.html

You need to enter the table name, where the users are stored in. In the "Attribute mapping" you may enter the fields:

username, phone, groups, mobile, email, surname, givenname, password

The attributes username and userid are required. All other attributes are optional. The password column in the attribute mapping holds a crypted password, where the first two bytes are the salt that was used to crypt the password. The tool *create-sqlidresolver-user.py* can be used to create crypted passwords.

E.g. if your user table has a column "loginname" and a primary key "id", you might at least enter:

```
{ "username":"loginname", "userid":"id"  }
```

In the field `database encoding` you enter the encoding of your database. This should be "utf-8" but could also be something like "latin1".

The field `Where statement` can be used to filter the users used for LinOTP. If you e.g. only want to use with an ID bigger than 5 you could enter a where statement like "id>5".

If your usernames or other fields contain non-ASCII characters you probably use another encoding. In this case you can enter the information, what encoding your database table uses.

Usually it might be latin1 or utf-8. You can also add additional parameters that get added to the connection string. The parameters very much depend on the used driver. If you are only using ASCII characters you will not have to bother with this setting. E.g. for a mysql database containing utf-8 tables and storing usernames with non-ASCII characters, you should add the parameters "use_unicode=1&charset=utf8".

After having entered all necessary values, you need to push the button "Apply" to store the changes. The changes will be sent to the server as soon as you push the button "OK" of the Configuration Dialog.

3.2.3 Flatfile UserIdResolver

The simplest userstore are flat files on the LinOTP server. These flat files can be used by configuring the Flatfile UserIdResolver. You only need to enter the filename of the file on the LinOTP server. The

default filename would be /etc/passwd. Other files need to follow the syntax of the passwd file, i.e. each line represents a user entry in each line, all fields are separated by a colon (:)

The fields represent the following

1. column: loginname (used as loginname by LinOTP)
2. column: (not used – usually this would contain the password)
3. column: numerical id (used as unique ID by LinOTP)
4. column: (not used – usually this would contain the groupid)
5. column: display name (used as displayname by LinOTP)
6. column: (not used – usually this would contain the home directory)
7. column: (not used – usually this would contain the login shell)

After having entered all necessary values, you need to push “Apply” to store the changes. The changes will be sent to the server as soon as you push the button “OK” of the Configuration Dialog.

3.3 Configuring Realms

Even if you do not run a complex scenario and you are not in the need of realms, you need at least configure one default realm, which will contain all your UserIdResolvers.

Note: If you will use only one realm, you should set this realm the default realm. Otherwise the management Web UI will not display the users of this realm.

To configure realms open *Config* → *Server Configuration* and go to the tab “Realms”. Using the button “Add Realm” you may add new realms. The name of a realm is unique and case-IN-sensitive and may contain the following characters:

- letters [a-z]
- digits [0-9]
- dash [-]

To add UserIdResolvers to a realm, select the previously defined UserIdResolver from the combobox on the right side and click “Add userstore to realm”. You may also remove the UserIdResolver from the realm again. Using the second combobox you may choose, which of your realms should be the default realm. For the idea behind the default realm and the UserIdResolvers as far as authentication is concerned read section A.16.2 on page 75. The changes get written to the server and get active when you push the button “OK”.

4 System config

Using the System Config you can define some of LinOTP's overall behavior. This is the view of the System Config of the management web interface. The native client looks rather the same.

`Default Max FailCount` The FailCounter is a counter per token, that counts the failed logon attempts. Here you can set, how often the user may attempt to logon with a token, before this token is locked. This is a default value. You may change this per token.

`Default Reset FailCount` If this is set to true (checked), a successful logon with a token will reset the Counter of the failed attempts to zero.

This is a default value. You may change this per token.

`Default Sync Window` For event based (HOTP) tokens this is the counter window, how many blank presses LinOTP will calculated further from its last known counter.

This is a default value. You may change this per token.

`Default OTP Length` This is the length of the OTP value. This is used to split the OTP value from the OTP PIN. This is necessary for all token types.

This is a default value. You may change this per token.

`Split At Sign` This determines, how the username is handled during the login process. If set to true (checked) the following will be done: If the username contains a "@", the username will be split into username and realm name. E.g. the username "user1@company2 (user1@company2)" will be split into

- username = user1
- realm = company2

If SplitAtSign is false (not checked), the username will be always take as it is. i.e. LinOTP will look for a user "user1@company2 (user1@company2)" following the default user resolving techniques.

`Return SAML Attributes`

Starting with version 2.4 LinOTP is capable of communicating with simpleSAMLphp via the LinOTP interface /validate/samlcheck. If this is true (checked) LinOTP will not only return the information if the user successfully authenticated but also return the user attributes:

- username
- surname
- given name
- phone
- mobile
- email

`Increase FailCounter on false PIN` LinOTP will split the OTP value and then compare the remaining password as PIN to the PINs of each token assigned to the user. If the PIN matches to a token, LinOTP will calculate the OTP value of this token and compare it to the given one. If the OTP values do not match, LinOTP will increase the FailCounter for this very token. If "Increase

FailCounter on false PIN" is set to true (checked) and the PIN does not match to any token at all, LinOTP will increase the FailCounter of all tokens.

If it is set to false (not checked) LinOTP will not increase any FailCounters.

`Prepend PIN`

If set to true (checked) the user needs to put the OTP PIN in front of the OTP value. (e.g. "mySecret647356"). If it is set to false (not checked) the user needs to put the OTP PIN behind the OTP value. (e.g. "647356mySecret").

`Auto resynchronize` If Auto resync is true (checked) LinOTP will work like this: If a token is out of sync, LinOTP will remember the given OTP value for this user and for this token. If the user logs on during the timeout time and provides another PIN and OTP value, LinOTP will try to resynchronize the token – identified by the OTP PIN – with these two OTP values.

Of course the two OTP values need to be consecutive values.

`Auto resynchronize timeout`

This is the time, how long LinOTP will remember the first given OTP value. I.e. this is the time window, in which the user needs to enter two consecutive OTP values.

`Pass on user not found` If LinOTP is not able to resolve the given username during the logon process, access will be granted.

Warning: Use this with caution and only if you know what you are doing!

`Pass on user no token`

If no token is assigned to the user, LinOTP will grant access to this user during the logon process.

Warning: Use this with caution and only if you know what you are doing!

`TOTP timestep`

TOTP tokens are time based OATH tokens defined in RFC 6238. The counter is calculated from the UNIX system time. The counter increases either all 30 or 60 seconds. So you need to enter 30 or 60 here.

This is a default value. You may change this per token.

`TOTP timeshift`

This is the drift of the LinOTP clock from the TOTP token clocks. Usually you should have no default drift, but have your LinOTP server clock synchronized using NTP. So this should be set to 0.

This is a default value. You may change this per token.

`TOTP timewindow`

LinOTP can calculate and compare the OTP values from some seconds before the current time and after the current time. A sensible value could 60 or 120, so that LinOTP will calculate and compare OTP values 1 or 2 minutes before the current time and after the current time.

This is a default value. You may change this per token.

`Selfservice portal: Display realmbox`

If this is true (checked) a dropdown box containing a list of all realms will be displayed on the logon page of the selfservice portal. If this is false (not checked), no logon box will be displayed. This way, you can hide the names of all realms from the users. The user then needs to log on by entering *username@realm*.

5 | Supported tokens

LinOTP supports a broad range of different tokens from different vendors.

HOTP

Supports any kind of RFC 4226 compliant tokens. LinOTP can import OATH-compliant key files according to RFC 6030. Additionally it can import SafeNet eToken PASS XML files and Feitian XML files. Furthermore the key can be entered manually during enrollment. Thus LinOTP supports:

- SafeNet eToken PASS
- Feitian C100
- Authenex A-Key V3.6
- Safeword Alpine
- Validustech BC-30, CR-1, PB-1
- Many different kinds of mobile Apps. See section *Recommended Mobile Apps* (page 18).

HMAC-SHA256

Supports HMAC-SHA256. Other than RFC 4226 which is based on HMAC-SHA1 LinOTP also supports HMAC-SHA256. This is used e.g. with newer SafeNet eToken PASS tokens.

TOTP

Supports any kind of TOTP compliant tokens. LinOTP can import OATH-compliant key files according to RFC 6030. Furthermore the key can be entered manually during enrollment. Thus LinOTP supports:

- SafeNet eToken PASS timebased
- Feitian C200
- Validustech BC-30, CR-1, PB-1

SafeNet eToken NG-OTP

Supports the eToken NG-OTP. LinOTP can generate the HMAC key on the eToken NG-OTP.

Yubico Yubikey

Supports the Yubikey I, Yubikey II and Yubikey NANO in OATH mode. LinOTP can generate the HMAC key on the Yubikey.

Also supports the Yubikeys as shipped by Yubico with the originial Algorithm, creating the 44 character long password. The authentication is then forwarded to the Yubico cloud authentication API.

Vasco

Supports DigiPass Tokens in RO (Response Only) mode like GO1, DP300, GO3, GO6.

mOTP

Supports mOTP. LinOTP supports the motp1 Algorithm. The mOTP key can be entered during enrollment. For recommended apps see section A.

Simple Pass

Supports the Simple Pass Token, which is a fixed password token without any moving factor.

Remote Token

Supports Remote Token, which forwards OTP requests to other LinOTP servers, either based on user assignment or simply based on the token serial number, thus enabling complex distributed setups.

RADIUS Token

Supports RADIUS Token, which forwards the authentication request of username and password/OTP to any given RADIUS server, thus enabling smooth migration scenarios.

SMS OTP

Supports SMS Token. LinOTP can enroll SMS Token, which will send OTP values via SMS to the given cell phone number of the assigned user.

Day OTP / Tagespasswort

Supports Tagespasswort Tokens. LinOTP can import key files or enroll Tagespasswort Tokens, which will changes their value once a day and thus enable the usage of OTP in Applications that do not provide any external authentication interface like e.g. RADIUS.

5.1 Recommended Mobile Apps

There are many different Apps that implement the HOTP and the motp algorithm and that can be used with LinOTP. But many of them are not very comfortable to enroll.

5.1.1 Recommended HOTP Apps

Apps for the iPhone

Other Apps can be used, but the secret often needs to be registered manually and typed into the selfservice portal.

Google Authenticator

- Pro: Supports several different accounts / tokens.
- Pro: The secret can not be displayed within the app.
- Pro: Very easy enrollment by scanning the QR code in the LinOTP selfservice portal from within the app.
- Cons: The token can not be password protected in the app.

OATH Token

- Pro: Supports several different accounts / tokens.
- Pro: Using lockdown mode the secret can not be displayed within the app.
- Pro: Rather easy enrollment by scanning the QR code.
- Cons: But the code needs to be scanned with an external app like “Red Laser”. The scanned link then needs to be deleted manually.
- Cons: The token can not be password protected in the app.

DS3 Token

- Pro: Token can be password protected within the app.
- Pro: The secret can not be displayed within the app.
- Cons: Difficult enrollment since the secret needs to be entered in the LinOTP selfservice portal manually.

- Cons: Does only support one account / token.

Apps for Android

Google Authenticator

- Pro: Supports several different accounts / tokens.
- Pro: The secret can not be displayed within the app.
- Pro: Very easy enrollment by scanning the QR code in the LinOTP selfservice portal from within the app.
- Cons: The token can not be password protected in the app.

Androidtoken

- Pro: Supports several different accounts / tokens.
- Pro: The secret can not be displayed within the app.
- Pro: Very easy enrollment by scanning the QR code in the LinOTP selfservice portal from within the app. This token uses the Google Authenticator URLs.
- Pro: The token can be password protected in the app.

5.1.2 Recommended mOTP Apps

Apps for iPhone

iOTP

- Pro: Supports several different accounts / tokens.
- Pro: The secret can not be displayed within the app after enrollment.
- Pro: Can send automatic support-request email to administrator.
- Cons: The secret needs to be entered in the LinOTP selfservice portal manually.

Asion mobile OTP

- Pro: The secret can not be displayed within the app after enrollment.
- Cons: Only supports one account / token.
- Cons: The secret needs to be entered in the LinOTP selfservice portal manually.

6 Managing Tokens

Now that you setup your management client and your LinOTP server, you may start to manage the tokens of your users.

6.1 Import tokens

When you are using preseeded hardware tokens, where the seed was implanted in the factory, you will also get a file, that contains the serial numbers of the tokens and the corresponding seeds.

Warning: Please assure that no unauthorized person gets possession on those seeds.

LinOTP can import man different seed files directly.

The GTK management client can import Aladdin/SafeNet XML files.

The Web management client can import

- Aladdin/SafeNet XML files,
- OATH compliant PSKC files,
- simple CSV files for OATH (HOTP and TOTP) tokens
- and Vasco DPX files.

Note: If some very special file format is not supported it can be converted before importing the data.

Note: SafeNet Tokens often get delivered with a .dat file. This file can be converted with a linotp tool `convert-token.pl`.

6.1.1 Importing tokens with the GTK client

To import a file in the GTK client do the following:

1. Click the button “import tokenfile” in the left lower corner of the main window.
2. A file choosing dialog will pop up. Choose the corresponding file.
3. The progress of the import will be displayed and the tokens will be listed in the tab “Token” of the main window.

6.1.2 Importing tokens with the Web UI

The Web management client can import more different token files.

From the dropdown menu you can choose, which type of token file you wish to import.

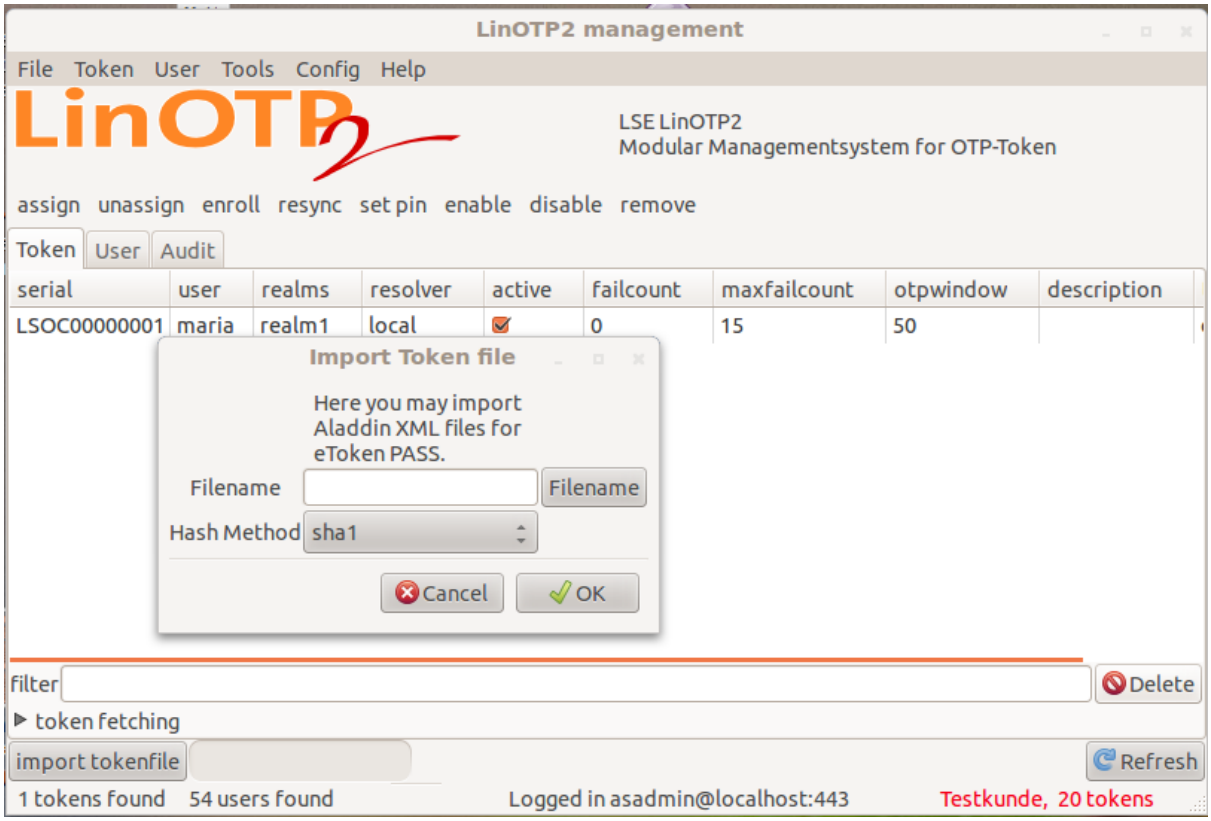


Figure 6.1: *Importing Aladdin XML files with the GTK client*

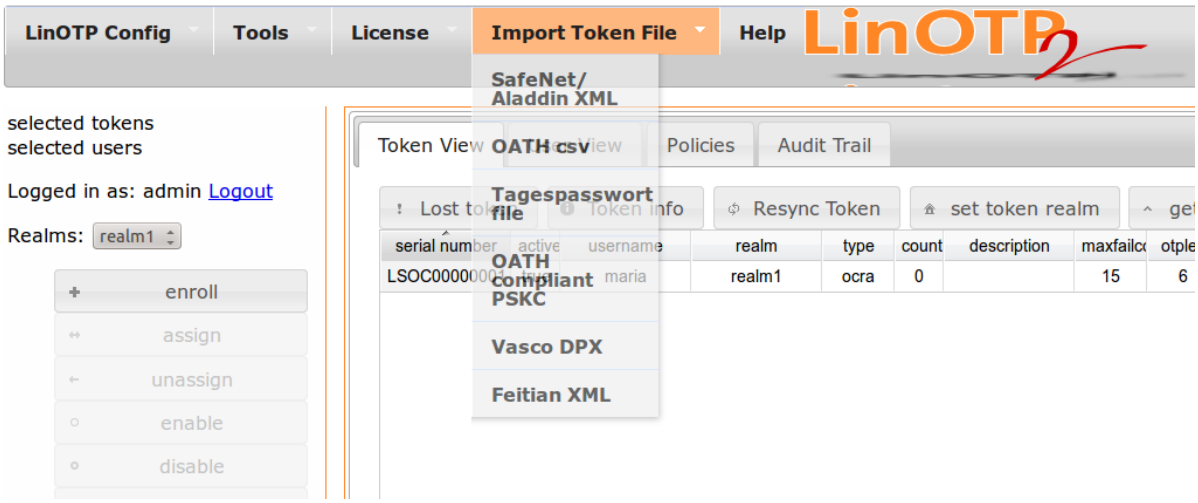
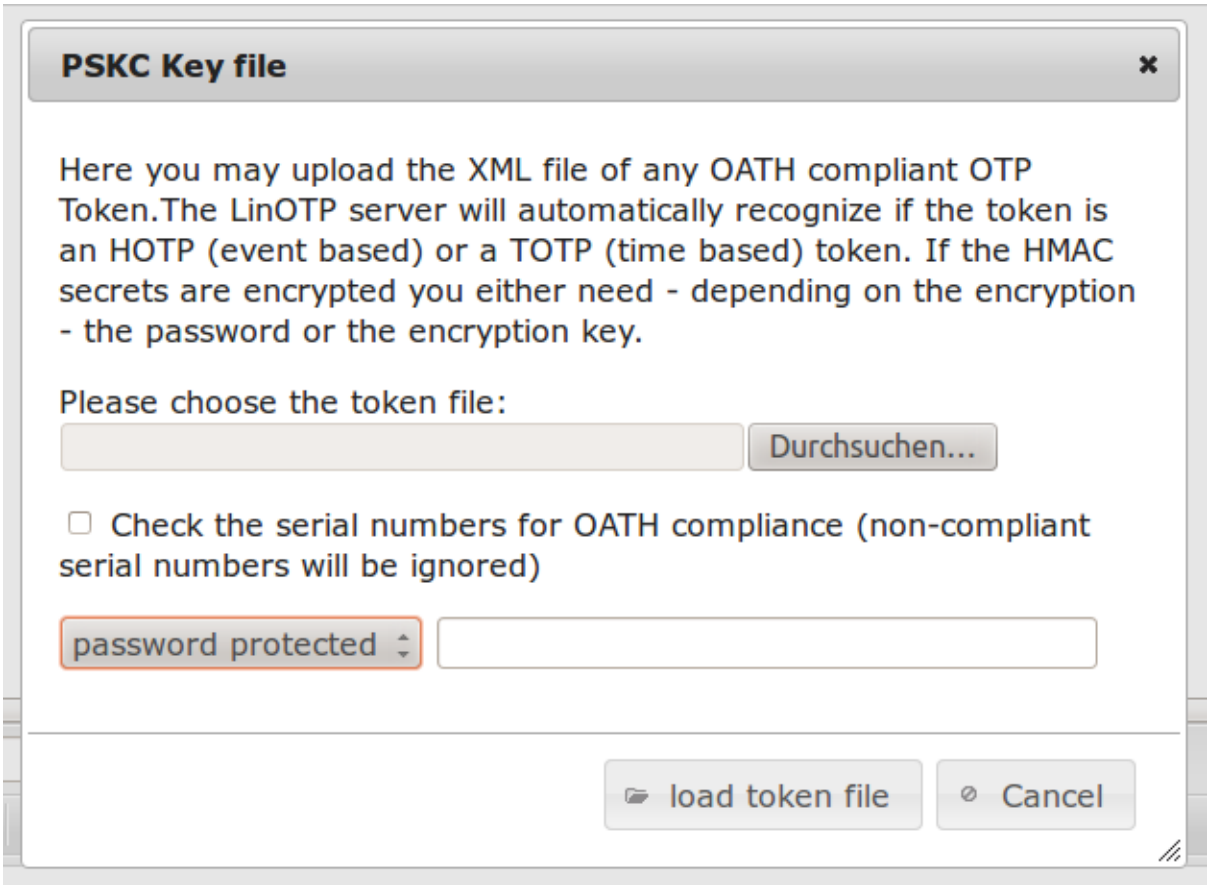


Figure 6.2: *Importing token files with the Web management client*

Importing PSKC files

PSKC (Portable Symmetric Key Container) is defined in RFC 6030 [1] and is used for OATH compliant tokens. OATH is the Initiative for Open Authentication where several vendors meet to use and define open standards to make strong authentication simpler and more compatible.

Note: If a token vendor claims to sell you OATH compliant tokens the vendor must deliver you a compatible PSKC file.



The PSKC file can contain HOTP or TOTP tokens and the seeds in the file can either be plain text or encrypted with a password or a preshared key.

The OATH standard also defines that the serial numbers of the tokens have to follow a certain naming scheme. Some vendors do not use this scheme, so you either check or uncheck the checkbox `Check the serial numbers for OATH compliance`.

If you check this checkbox, tokens with non-compliant serial number will not be imported.

In the dropdown box you can choose if the seeds in the PSKC file are `plain value`, `password protected` or encrypted with a `preshared key`. An input field will appear, where you can enter the password or the preshared key.

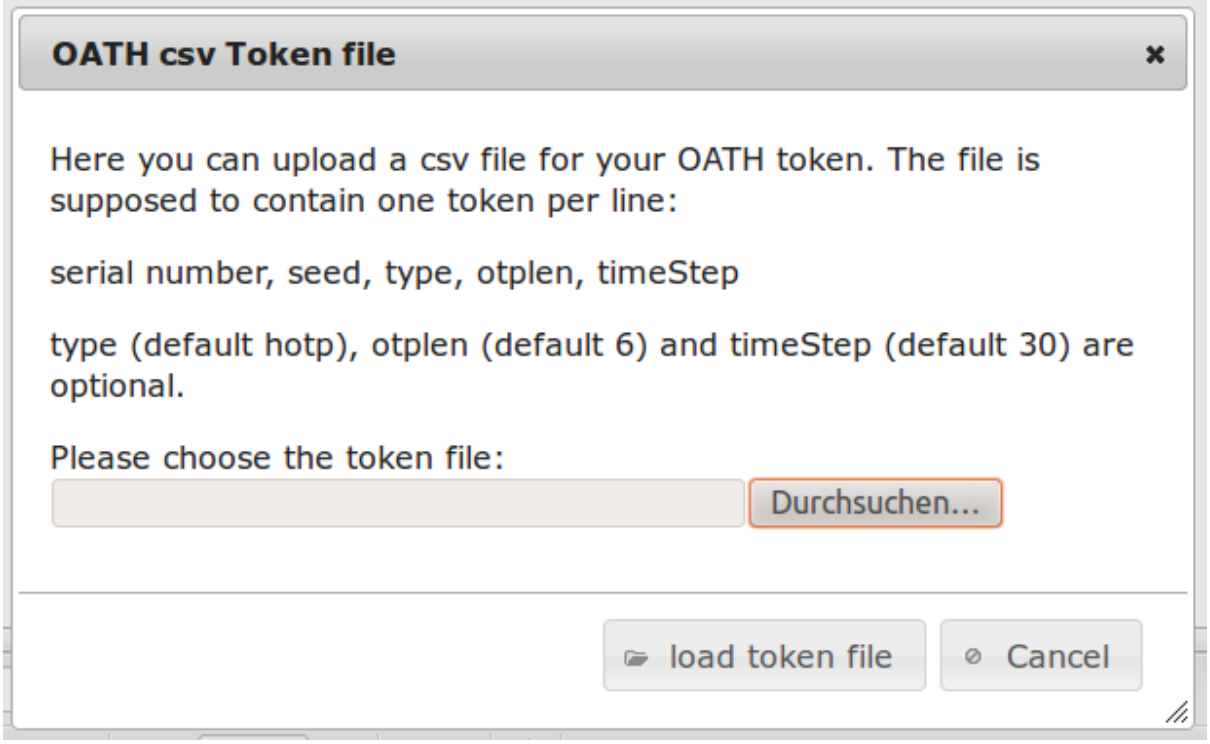
Importing OATH CSV files

This import dialog can be used to import OATH tokens, that do not provide a PSKC file.

The seeds can be stored in a simple comma separated file.

The fields in the CSV file need to be

[1] http://tools.ietf.org/html/rfc6030



- serial number,
- seed,
- type (optional, default=hotp),
- OTP length (optional, default=6),
- time step (optional, default=30).

Note: Depending on the length of the seed the token is either imported as an HMAC-SHA1 (40 characters = 160 bit) or an HMAC-SHA256 (64 characters = 256 bit) token.

The file can contain different types of tokens at the same time. So a valid file might look like this:

```
TS000001, 1f6aeda29fed39a8e2c3fe45c954d9ba93a14af4
TS000002, e3a391658226f63153443bb03a365eb962e1775b, hotp, 8
TS000003, bd15fb2b2c84a3ce56670fe0062b7369a0b8f4d4, totp, 6
TS000004, 881c7498360553b0e51a677ad7daa41b3b390ad5, totp, 8, 60
TS000005, 0eb6597f402151f97726208dc7e94bd541ff56b5a3ff63003c8ff0b6049185d7
```

The token TS000001 will be imported as a SHA1 HOTP token with 6 digits.

The token TS000002 will be imported as a SHA1 HOTP token with 8 digits.

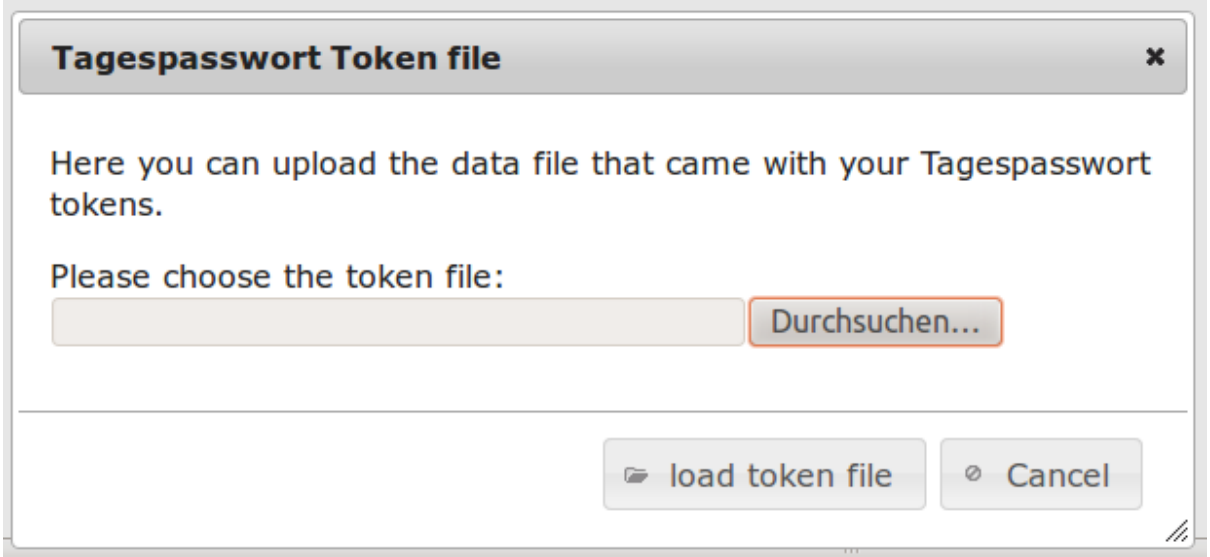
The token TS000003 will be imported as a SHA1 TOTP token with 6 digits and a time step of 30 seconds.

The token TS000004 will be imported as a SHA1 TOTP token with 8 digits and a time step of 60 seconds.

The token TS000005 will be imported as a SHA256 HOTP token with 6 digits.

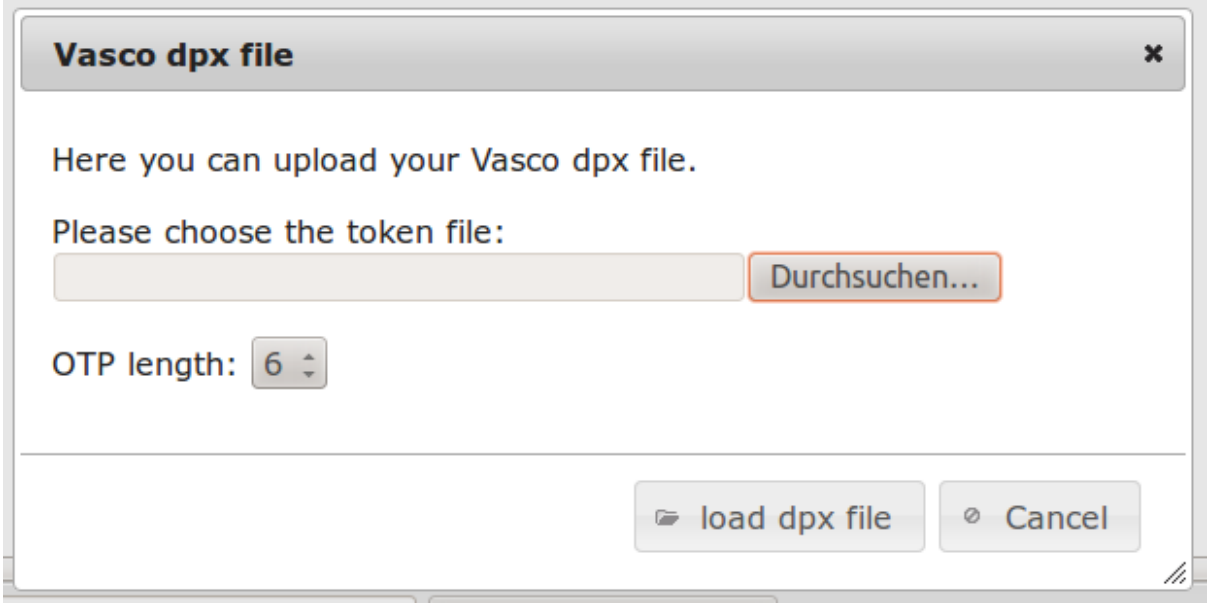
Importing Tagespasswort files

The *Tagespasswort* token is a token that displays a password valid for one day. I.e. the user can use this password several times to authenticate during this day. It will change on the next day.



The file format is fairly simple and just consist of one token per line with the serial number and the seed divided by white spaces.

Import Vasco DPX files



You can import the proprietary Vasco DPX file for the Vasco Digipass tokens.

To import the DPX file and use the Vasco tokens you need to license the Vasco vacman controller library and install it on the LinOTP server.

- Install the vacman controller library *aal2sdk* on your system.
- Locate the shared object file, it might reside at */opt/vasco/VACMAN_Controller-3.11.2/lib/libaal2sdk.so*.
- Now you need to set the variable *linotpImport.vasco_dll=/opt/vasco/VACMAN_Controller-3.11.2/lib/libaal2sdk.so* in the DEFAULT section in your *linotp.ini* file.
- Restart your LinOTP.

6.2 Viewing users in certain realms

You can group users into realms. In the GTK client and in the Web UI you can choose, which users from which realm you want to view.

If the administrator got restrictive policies to only manage tokens in certain realms (see chapter *Policies* (page 47)) he might need to select this very realm to display the users in it.

6.2.1 Viewing users in GTK client

By default you will see *all* users of all realms in the GTK client user view.

If you want to only view users from on realm, this realm of the user can be selected on the users tab under `Userfetching`.

Choose a realm and press `Refresh`.

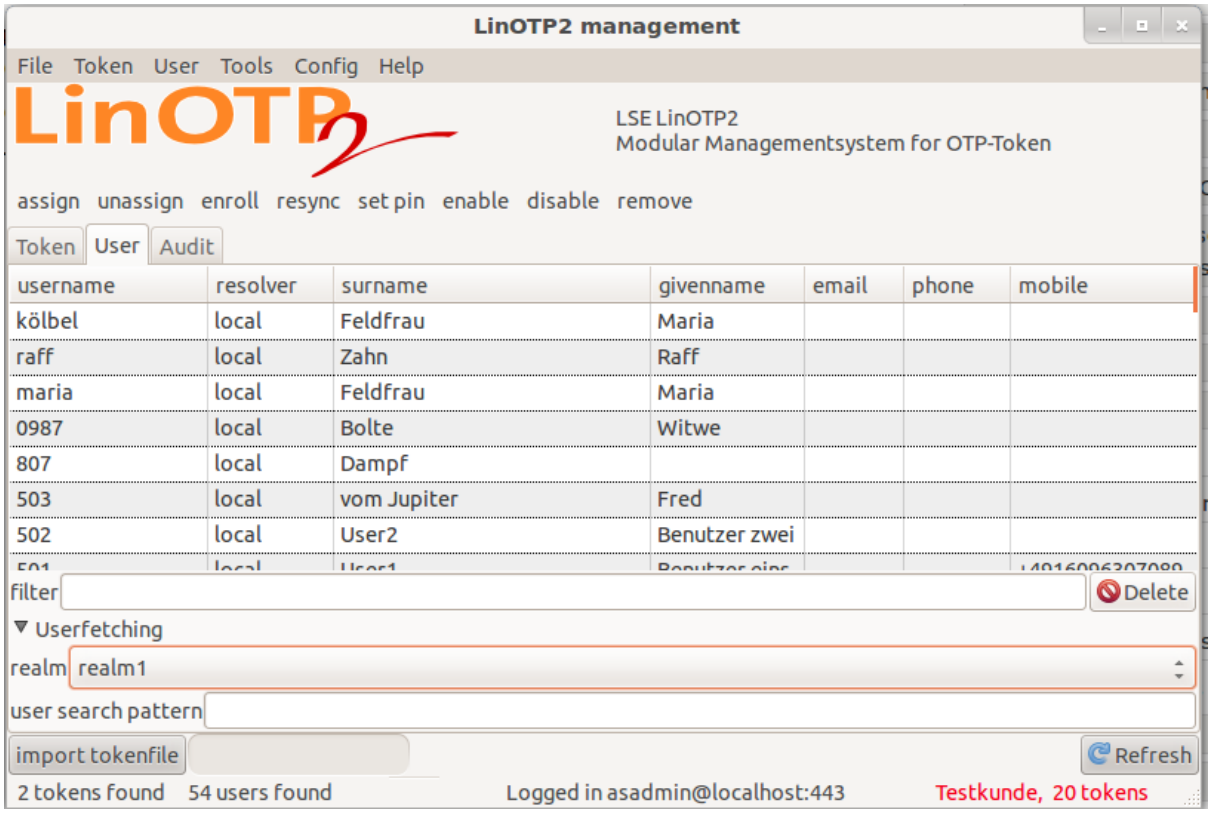


Figure 6.3: *Use userfetching to only select users from a certain realm*

6.2.2 Viewing users in the Web UI

Viewing of the users in the Web UI works a bit different.

On the left side you need to choose, which user realm you want to view. As soon as you change the realm on the left sidebar, the users from the newly selected realm will be displayed in the tab `User View`

You can use the pagination arrows to step through the users in this realm.

You can search by entering a search pattern in the `Find` field below the user list.

You can use the wildcard *. Entering `*a*` will find the users “maria” and “raff”. Entering `50*` will find the users 501, 502 and 503.

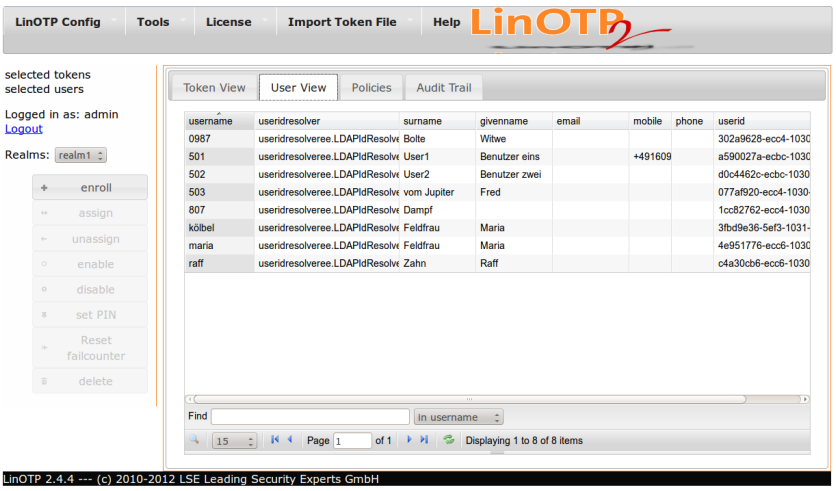


Figure 6.4: *Viewing users in the web UI*

When having entered the search pattern hit `Enter` to start the search.

6.3 Viewing tokens

You can view tokens by either providing a serial number or by providing a username.

If you are viewing tokens for user “a” LinOTP will try to find all tokens of user “a”.

If user “a” does not exist, LinOTP will try to find all tokens of users, that contain the substring “a” like the users “raff” and “maria”.

Warning: This behaviour can be very time consuming.

Note: This behaviour is deprecated and might be changed in the next release to use wildcards to find tokens by a username substring.

6.4 Assign tokens

In section *Import tokens* (page 21) you saw how to import token files.

The imported tokens are now available in LinOTP and you may now assign a token to a user.

When assigning a token to a user, the token will be put into the realm of this very user.

A restricted realm administrator, who is only allowed to manage tokens in certain realms, will not see tokens, that are not within his realm. So in this case the token first needs to be put into the realm.

For how to do this see section *Manage tokens* (page 40).

6.4.1 Multiple tokens for a user

It is possible to assign several tokens to a single user. Although the token is identified by the OTP PIN even each token may have the same OTP PIN. In this case, the OTP value is checked with every token, that belongs to the user and matches the OTP PIN.

6.4.2 Assign tokens using the GTK client

1. To find the token, you want to assign, you may enter a search term in the input line “filter”. The token is searched while you are typing.
2. Then choose one ore more tokens by left-clicking the tokens.
3. You may now assign the tokens by either right-clicking, clicking on button “assign” or choosing “assign” from the menu “Token”.
4. The dialog “assign token” pops up.
5. You may again search for the user by starting to type the username into the “User” field. The list of the users in the drop down right above the user field will be filtered accordingly.

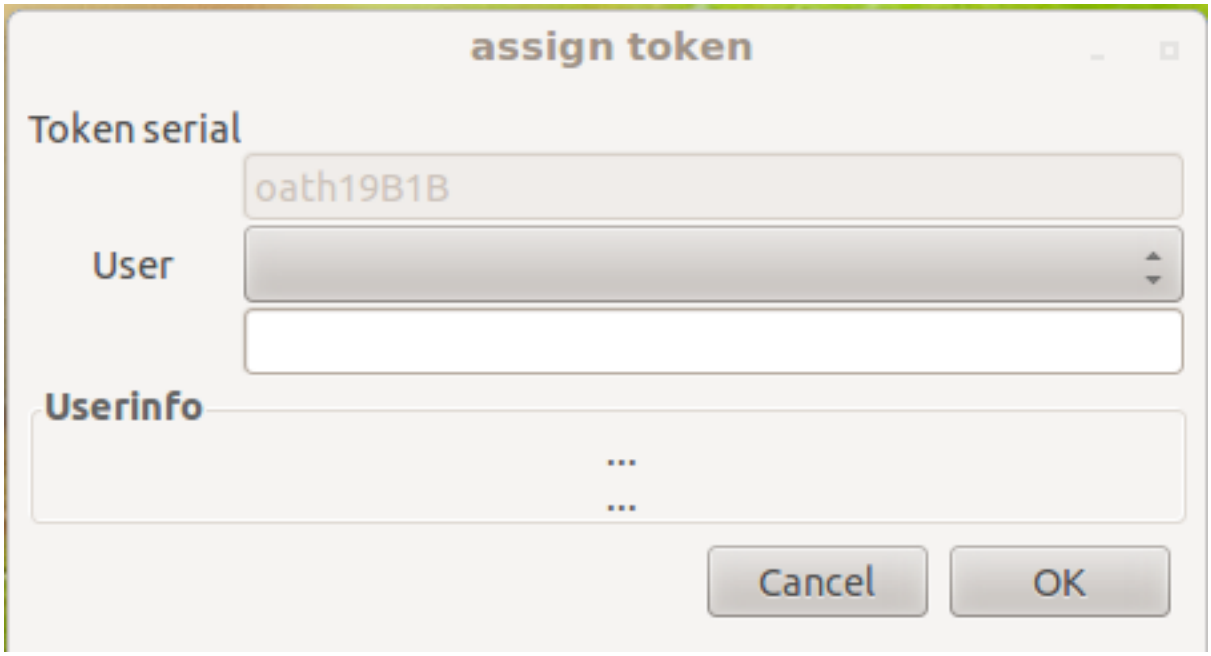


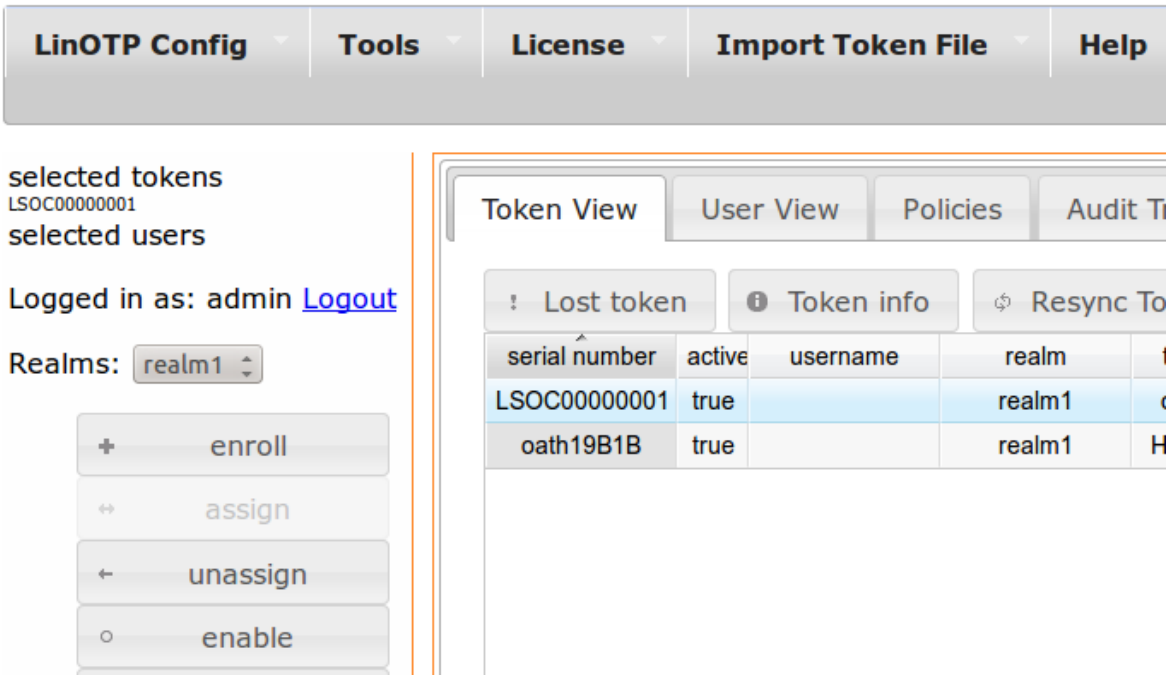
Figure 6.5: *assign token*

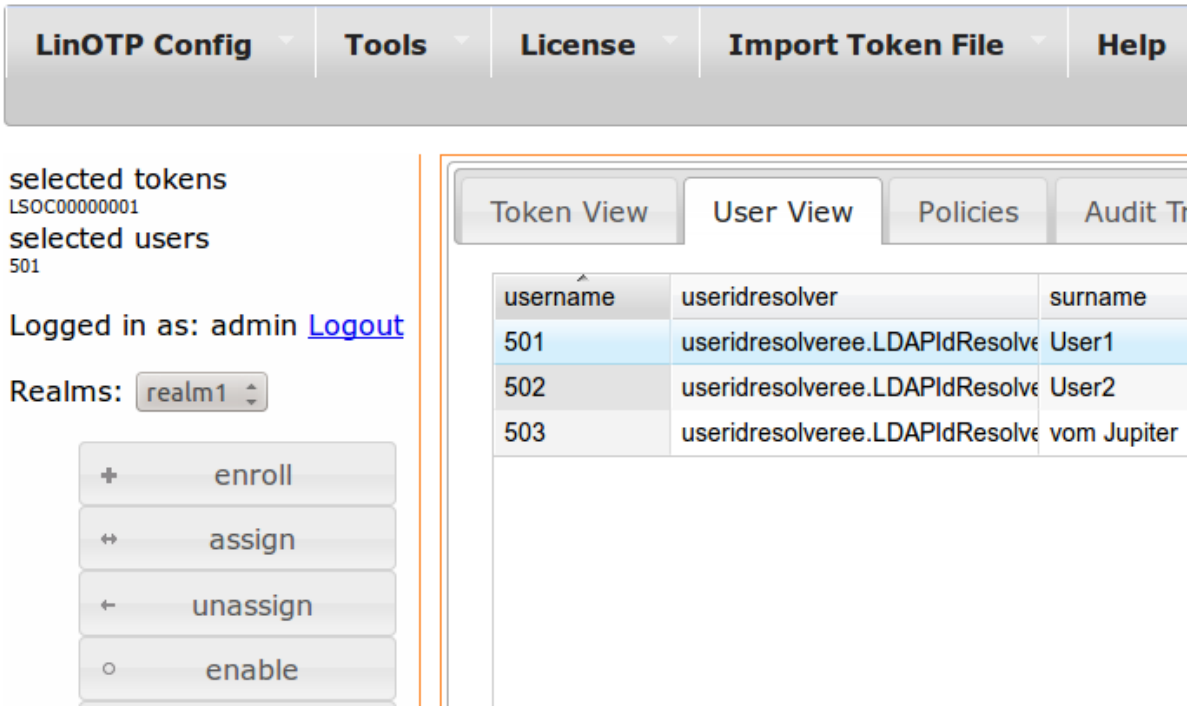
6. Now choose the user you want to assign the tokens to in the drop down list. The UserIdResolver, where the user was found, is also displayed. Thus you can distinguish between users with the same name residing in different user databases. Click OK. The tokens will now be connected (assigned) to the user selected in the drop down.
7. If you want to see all tokens again, just delete the filter entry.
8. If you only want to see the tokens, that are assigned to the user “maria”, enter “maria” into the filter entry.

6.4.3 Assign tokens using the Web UI

1. To assign a token using the Web UI you need to select the token, you want to assign.

   The tokens you selected are displayed on the left sidebar under “selected tokens”.
2. Now change to the “User View” and select the one user you want the token assign to. The name of the selected user is displayed on the left sidebar under “selected user”.
3. Now press the button `assign` on the left sidebar.
4. After assigning the token you may directly enter the OTP PIN for this token.


LinOTP Config
Tools
License
Import Token File
Help
selected tokens
LSOC00000001
selected users
Logged in as: admin Logout
Realms: realm1
enroll
assign
unassign
enable
Token View
User View
Policies
Lost token
Token info
Resync To
serial number
active
username
realm
LSOC00000001
true
realm1
oath19B1B
true
realm1


LinOTP Config
Tools
License
Import Token File
Help
selected tokens
LSOC00000001
selected users
501
Logged in as: admin Logout
Realms: realm1
enroll
assign
unassign
enable
Token View
User View
Policies
username
useridresolver
surname
501
useridresolveree.LDAPIdResolve
User1
502
useridresolveree.LDAPIdResolve
User2
503
useridresolveree.LDAPIdResolve
vom Jupiter

6.5 Set OTP PIN

When authenticating a user has to enter the OTP value displayed by the token plus an additional OTP PIN, which he needs to memorize thus doing a two factor authentication of the OTP value created by the token (possession) and the OTP PIN (knowledge).

The OTP PIN may either be in front or behind the OTP value. This can be configured in the system settings. See section *System config* (page 15) on how to do this.

The OTP PIN may either be an additional PIN stored with the token in the LinOTP token database or it may be the password taken from user store. See the policy section *OTP PIN variants* (page 57).

You may set an OTP PIN for all selected tokens.

Setting the OTP PIN works rather the same in the GTK client and in the Web UI:

1. Select one ore more tokens.
2. Choose “Select OTP PIN”.
3. In the popup dialog enter the OTP PIN.

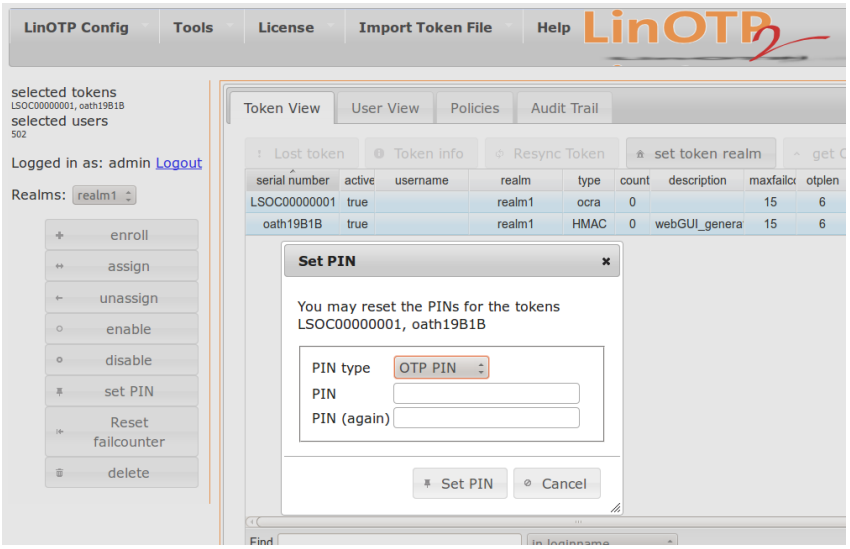


Figure 6.6: *Setting the OTP PIN for two tokens*

6.6 Enrolling tokens

LinOTP is able to enroll several different tokens. These can be:

- eToken NG-OTP,
- mOTP token,
- HOTP token,
- LSE simple Pass token,
- SMS OTP / Mobile TAN.
- Remote Token,
- RADIUS Token,
- Yubikeys.

6.6.1 Enroll eToken NG-OTP

Note: An eToken NG-OTP can only be enrolled using the GTK client or the command line client.

An eToken NG-OTP is a USB based token, that does not have a preinstalled HMAC Key. The secret HMAC key is generated by the management client and installed to the token and stored in the token database without any person seeing this HMAC Key.

For enrolling eToken NG-OTP you need to have the Aladdin/SafeNet PKI Client installed.

1. Push the button `enroll token`.
2. You may choose a user as you did during token assignment, but you may also leave the user blank. This enables you to mass-enroll eToken NG-OTP tokens and assign the tokens to users later.

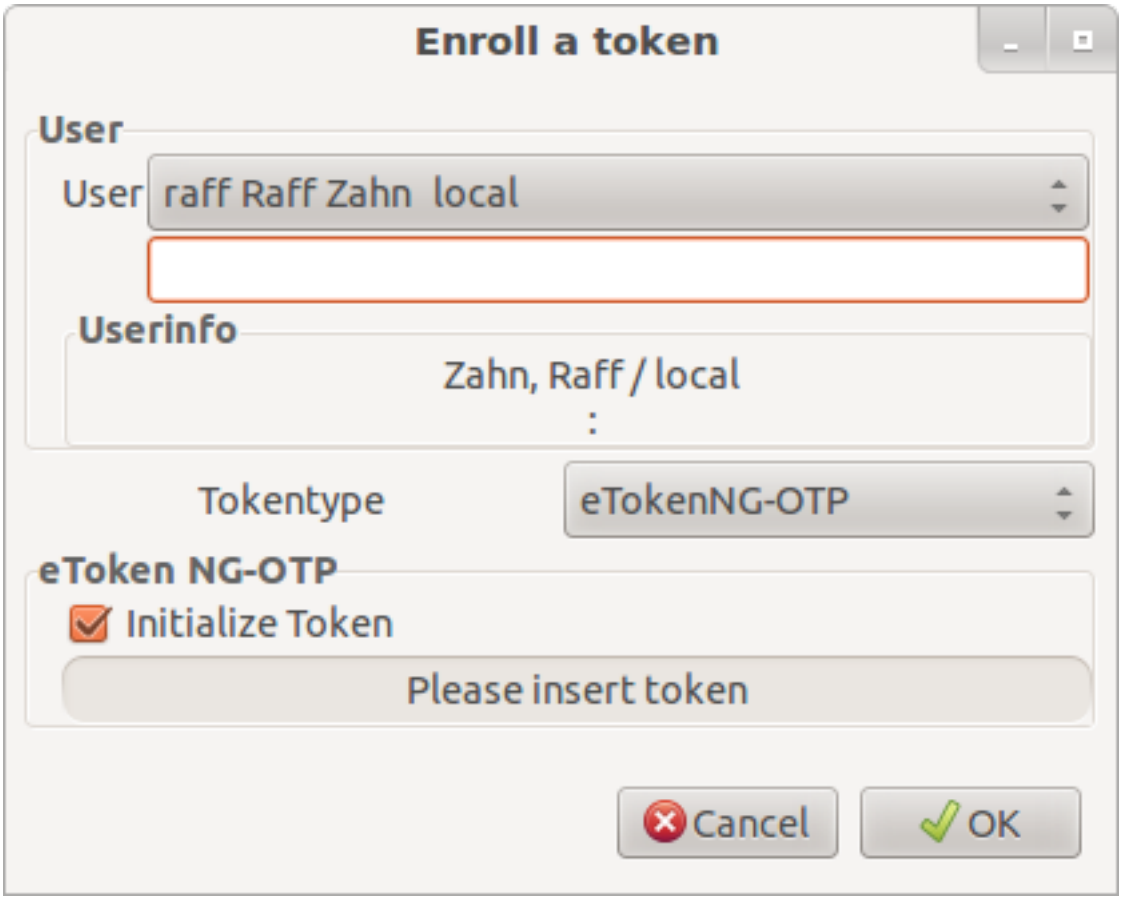


Figure 6.7: *Enroll an eToken NG-OTP using the GTK client*

3. Leave the `Tokentype` as it is.
4. Plugin one eToken NG-OTP. Be sure to have no other eTokens connected!
5. Push the button `OK`.
6. After enrolling the token you may also want to set the OTP PIN.

Note: During enrollment the eToken NG-OTP gets initialized.

Warning: All data on the token will be lost!

In the management client configuration you may choose, if the eToken NG-OTP should be initialized with random User PINs and SO PINs or if you want to set the PINs manually. The SO PIN is stored in the tokenbase for later use, for example resetting the User PIN.

6.6.2 Enroll mOTP Token

LinOTP is also capable of using other OTP algorithms and thus supports many different tokens. So it is also possible to enroll mOTP Tokens which you can download at http://motp.sourceforge.net. These are time based open source OTP tokens that can be installed on a mobile phone. mOTP token apps exist for Android phones, the iPhone and also for older feature phones. You can find recommended mOTP apps for Android and iPhone in the section *Recommended Mobile Apps* (page 18).

Please refer to the project website on how to install the motp on your feature phone.

As to the app for the feature phone you may initialize the motp application at any time anew. Initializing motp means, that a new OTP Key – in this case it is called Init-Key – is created on the mobile phone. This can be done by entering '0000' when the motp application asks 'Enter PIN:'. When you have installed the motp to your mobile or to the mobiles of all your users, each mobile needs to create its secret key.

1. In the LinOTP management client press the button "enroll".
2. Choose the tokentype "mOTP".
3. You need to enter the OTP Key, that was generated on the mobile phone.
4. The user needs to make up a 4 digit passphrase, that he enters each time, when he wants to generate an OTP value on his mobile.

   Note: This is the mOTP PIN. The mOTP PIN is the PIN that gets entered on the phone and is also used to calculate the OTP value. When you enter a wrong mOTP PIN on the phone, the phone will calculate a wrong OTP value. This must not be confused with the OTP PIN.

   ---

5. You may now press "OK" and the management client will generate a serial number for the token for you.
6. If you need to change the mOTP PIN later, you can do this in the GTK client by choosing an mOTP token and right-clicking the token and select "set mOTP pin". In the Web UI click "set PIN" a and choose "mOTP PIN".
7. You may also want to set an "OTP PIN", that the user needs to enter in front of the generated OTP value when he wants to authenticate.

The OTP value of mOTP tokens are validated by the epoch time. You may verify the epoch time on the LinOTP server to compare it to the time on the mobile phones using the command:

```
date +%s
1268661055
```

6.6.3 Enroll HOTP / OATH Token

You may realize, that there are several softtokens (e.g. for mobile phones) around that use the HOTP algorithm defined in RFC4226. Some of these tokens generate the HMAC secret key during an initialization process. For these tokens, you may choose to enroll "HOTP token" and than enter the HMAC key in this dialog.

1. Choose tokentype as "OATH"
2. Enter the secret, that was generated by the token.

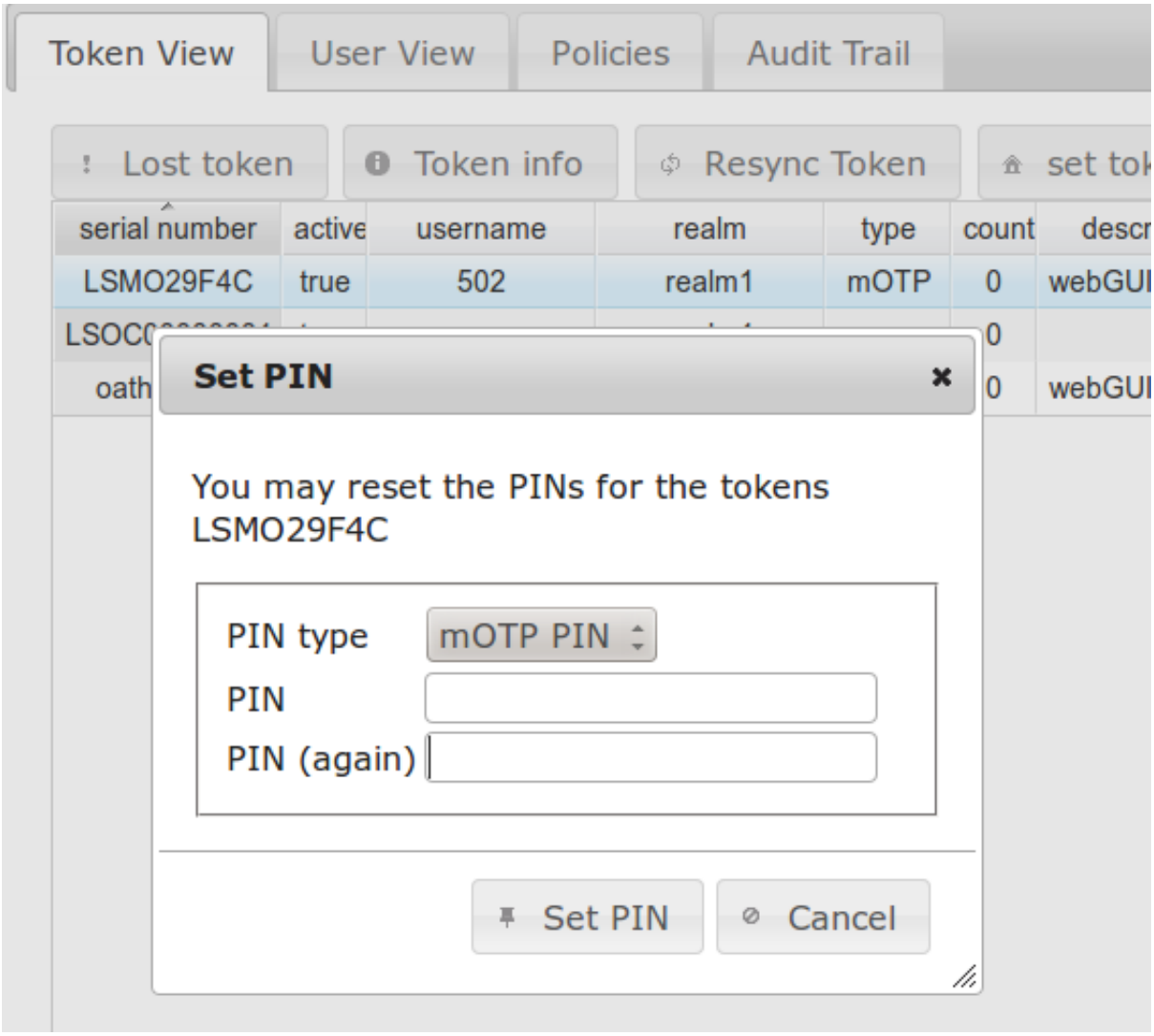


Figure 6.8: *Setting the mOTP PIN in the Web UI*

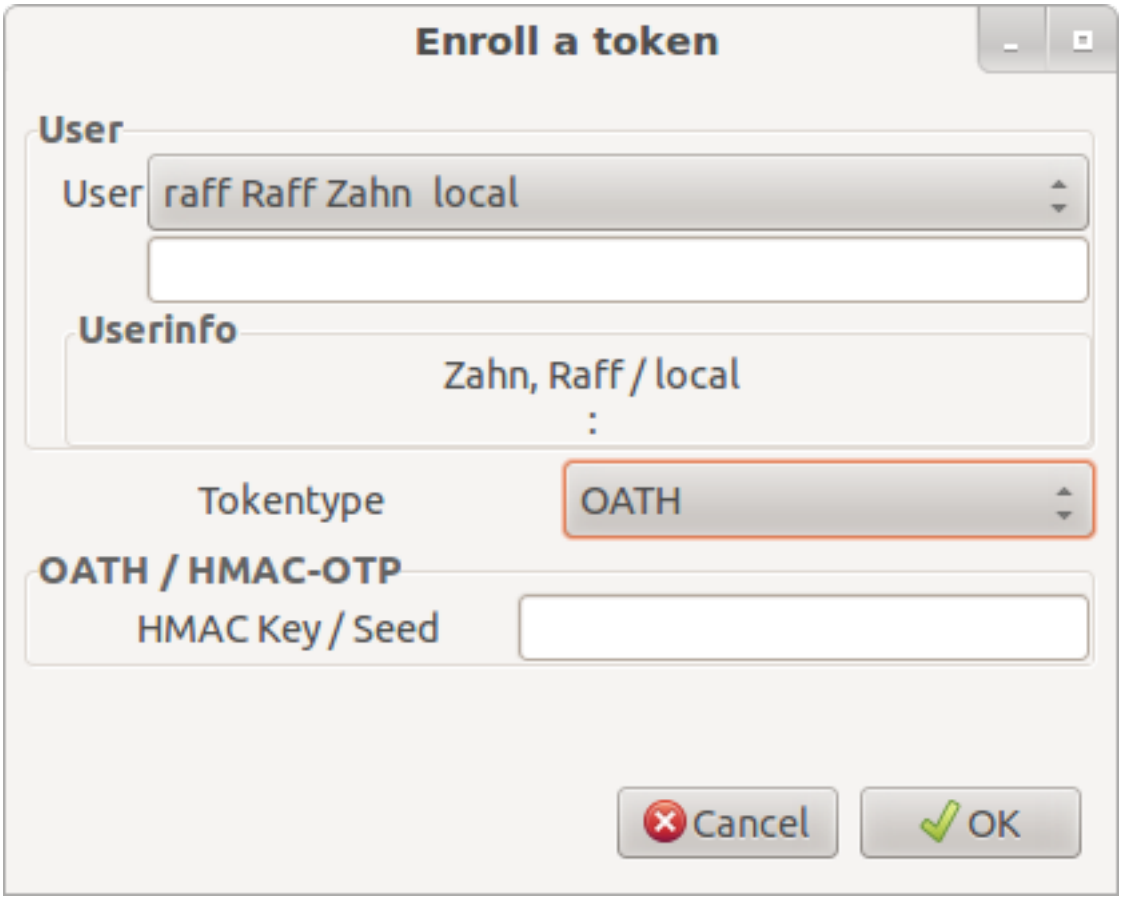


Figure 6.9: *Enroll an OATH token with the GTK client*

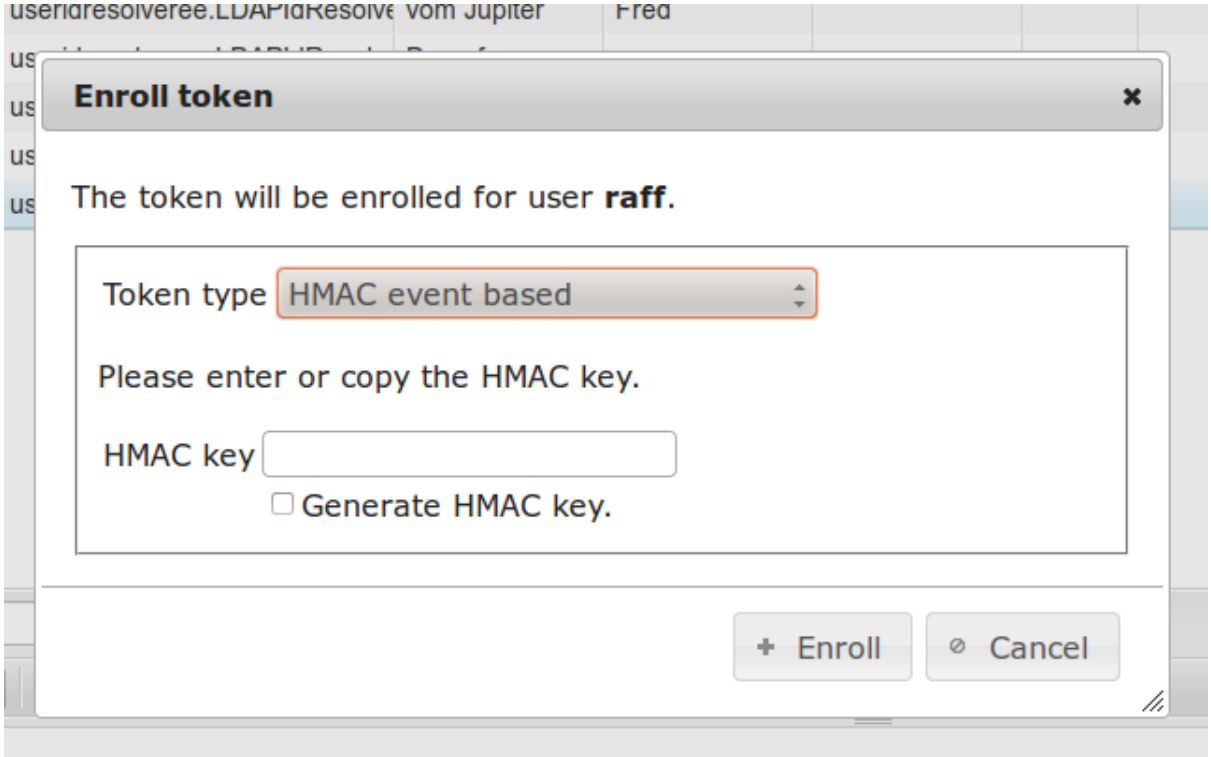


Figure 6.10: *Enroll an OATH token with the Web UI*

This looks pretty much the same in the GTK client and in the Web UI.

As a matter of fact the Web UI offers an additional functionality to enroll a soft OATH token. The Web UI can have the LinOTP server generate the HMAC seed. Some Apps like the Google Authenticator, the Android token or OATH token can import this seed using a QR Code.

The LinOTP server can generate such a QR code.



Figure 6.11: *Enroll a Google Authenticator using a QR code*

6.6.4 Enroll LSE Simple Pass Token

In some cases it might be necessary that users authenticate without a one time password, but rather with a fixed password. This may be, if your authentication client is not capable of routing users to OTP authentication and other users to password authentication or in cases, where a user may have lost his OTP generator.

To enroll a Simple Pass token

1. choose the tokentype “SPass”
2. push the button “OK”.

If you do not set an OTP PIN now, the user will get successfully authenticated ALWAYS! So you now should set an OTP PIN for this token. The user may now authenticate against LinOTP by providing only this PIN.

6.6.5 Enroll SMS OTP / Mobile TAN

To enroll an SMS OTP token

1. choose the tokentype “SMS”.
2. If you selected a user and this user has a mobile phone number defined in the user store, this phone number will be automatically entered in the phone number field. You may of course change this.
3. Press OK and the token with this mobile phone number is created and assigned to the selected user.

6.6.6 Enroll Remote Token

A Remote Token forwards the authentication request to another LinOTP server. So the calculation of the One Time Password is not done on the first LinOTP server, where the User is authenticated but on the remote LinOTP server. The Remote Token can be defined this way, that only a serial number and the OTP value is forwarded to the remote server. This token with this serial number needs to be located on the remote server of course.

Thus, a remote server can hold all OTP tokens without knowing any users and without having any tokens assigned to any users. On such a remote server not even a UserIdResolver needs to be defined.

The authenticating LinOTP (first) server will do no OTP calculation and thus will not know any secret keys (HMAC keys) since it forwards all requests to the remote server. To enroll a Remote Token just choose enroll and then the token type “Remote Token”.

Then you need to enter the remote LinOTP server and the serial number of the token on the remote server.

Alternatively you can specify a user on the remote LinOTP server, if the token is assigned to a user.

Note: If you specify “https://localhost” as the remote server, you can assign one single physical token to several users, by pointing the Remote token to a local token of another user.

You may choose if the checking of the PIN is done locally or remotely.

`Check PIN remotely` Do not set a PIN on the local LinOTP server. The entered credentials will be completely forwarded to the remote LinOTP server. The PIN that was set on the remote server will be used to authenticate the user.

`Check PIN locally` In this case, the PIN on the local LinOTP server is verified. If the PIN on the local server matches, the remaining part of the credentials will be sent to the remote server. On the remote server NO PIN should be set for the token, otherwise authentication might fail. In this scenario a remote server can hold one physical OTP token with no PIN, while all the Remote Token on the local machine get individual PINs and can be assigned to different users.

6.6.7 Enroll RADIUS Token

A RADIUS Token can forward the authentication request to another RADIUS server. This can be any kind of RADIUS server.

Note: The RADIUS token can be well used for migrating existing RADIUS scenarios.

A RADIUS Token consists of

- the RADIUS server name
- plus the port,

- the username on the RADIUS server and
- the RADIUS shared secret.

You may also define, if the OTP PIN is checked locally on the LinOTP server or if the complete password is forwarded to the RADIUS server.

Choose “enroll” and then choose the token type “RADIUS Token”. Enter the RADIUS server name including the port and the other required information.

`Check PIN locally` If the OTP PIN is checked locally, you also need to specify the length of the potential OTP value, i.e. how many characters of the password will be split and sent to the RADIUS server.

`Check PIN on RADIUS server` If you choose to check the PIN on RADIUS server the complete credential will be sent to the RADIUS server.

6.6.8 Enroll QR-TAN Token

A QR-TAN Token is a special OCRA Token that can be enrolled via a QR code process and where the challenge is also transmitted via a QR code. You can either start the QR enrollment via the API (/admin/init) or you can start the enrollment of the QR Token via the management web UI.

Note: Please note, that you need the LSE QR-TAN App which is available for iOS and Android.

Select a user and press the button “Enroll Token”.

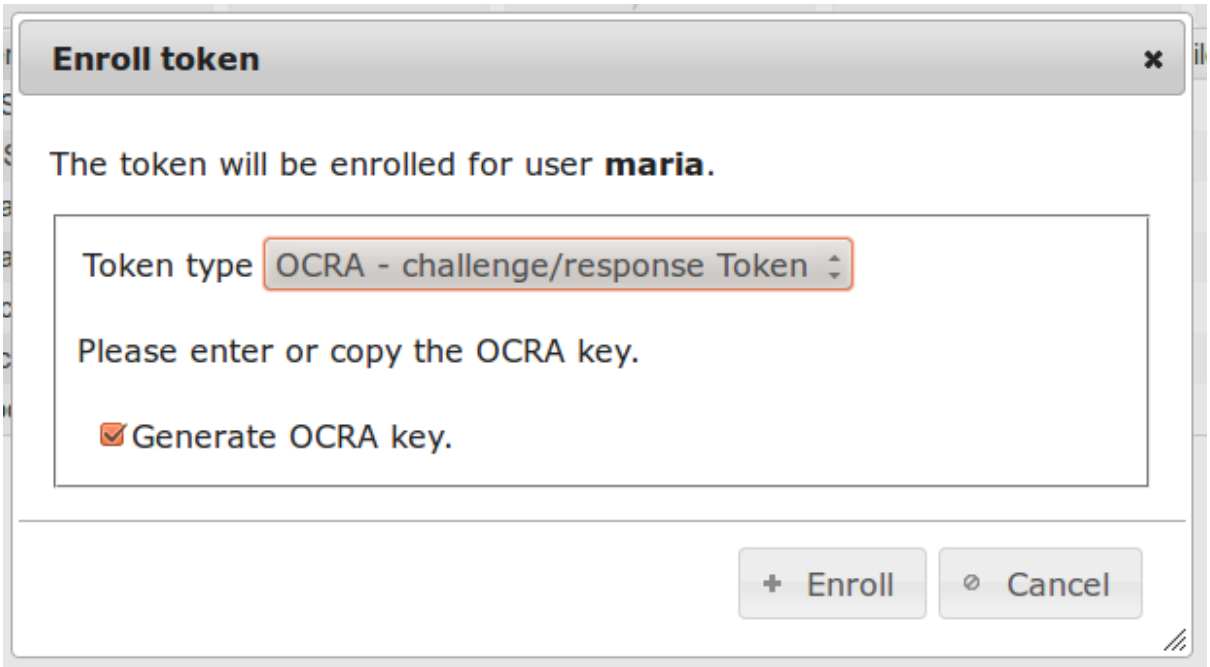


Figure 6.12: *Start enrolling a QR Token*

The displayed QR code needs to be scanned with the users smartphone. Alternatively you can print the QR code and hand it over to the user.

In a second step the user needs to login to the selfservice portal and go to the tab .... and select the corresponding token. The user needs to enter the activation code, which is displayed on his smartphone and then needs to scan the second QR code, which is displayed in the selfservice portal.

Now the QR Token is enrolled and ready for use.

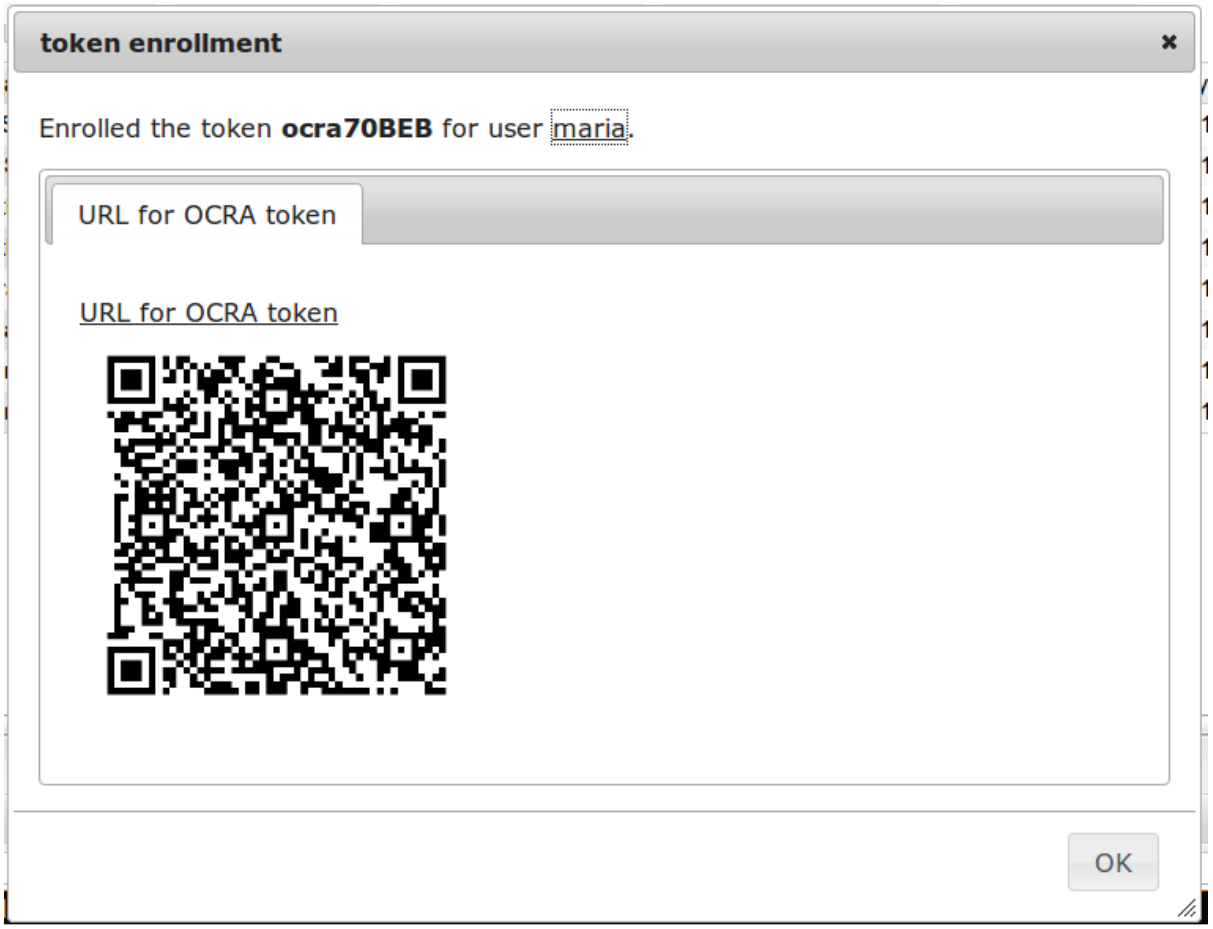


Figure 6.13: *LinOTP generated the initial shared secret and the QR code*



Figure 6.14: *Finalize QR Token enrollment*

6.7 Manage tokens

You can now manage all your existing tokens. This includes the already described processes of enrolling and assigning tokens.

6.7.1 Unassign

If a user does not need his token anymore, the token can be unassigned from the user.

The token will not be assigned to the user anymore and the OTP PIN of the token gets cleared.

6.7.2 Remove

If a token is completely lost or is not needed anymore, the token can be removed from the token database.

Note: All entries about this token will remain in the audit trail.

6.7.3 Disable

If a user should not use the token for authenticating – maybe because he forgot the token somewhere – the token can be disabled. This is reflected by the column “active”.

In the GTK client you can disable all tokens of a user by choosing the user in the users tab, right-click and choose “disable”. All tokens that are assigned to the user are disabled.

Note: Later you can enable the token again.

6.7.4 Reset failcounter

The failcounter reflects how often the user failed to authenticate with this token.

The failcounter is increased under the following conditions. The last 6 or 8 characters are removed from the provided password. This is assumed to be the OTP value. The remaining leading characters are interpreted as the fixed OTP password/ OTP PIN. If LinOTP finds a token, which matches this OTP PIN but not the OTP values, the failcounter for this very token is increased. If LinOTP does not find a token, which matches the OTP password, the failcounters for all tokens assigned to this user are increased. If the failcounter reaches the the value of max failcounter, login with this token is not possible anymore and the failcounter will not be increased any further. If a user authenticated successfully the failcounter for this token is reset to zero.

To reset the failcounter manually, select a token in the token view, right-click [2] and choose “reset failcount”. You may also select several tokens to reset the failcounter.

6.7.5 Resync token

Event based tokens like the HOTP token work this way, that the token has a counter which is increased on every creation of every new OTP value. The LinOTP server also tracks this counter.

The LinOTP server has a “count window” defined per token. The server will try to validate the entered OTP value within this count window.

[2] The right click works in the GTK client. In the Web UI you need to choose “reset failcounter” from the left sidebar.

Logically the counter can run out of sync, when the new OTP values are not passed to the LinOTP server for authentication. When the user creates many new OTP values but only authenticates with the 20th OTP value while the count window on the server is only 10, then the server will not be able to find this OTP value within the count window and will fail to validate the OTP value.

The token is out of sync and needs to be resynced.

Note: You can also use the autosync function described in section *System config* (page 15).

To resync the token manually

- select a token
- choose “Resync Token”
- and enter two consecutive OTP values.

6.8 Set token realm

Each token may be in several realms at the same time.

When a token is assigned to a user, this token also gets put into the realm of the user.

But you can also put the token into several realms, so that realm administrators (see *Admin Policies* (page 50)) can see those tokens.

Imagine a scenario where you have a LinOTP installation with several realms. Each realm is managed by a dedicated realm administrator. Now you can import many tokens. By putting some of the tokens into one realm and other tokens into another realm you can configure which unassigned tokens are available in which realm.

To put tokens into a realm, you can select several tokens and choose “set token realm”.

6.9 Token info

When you select one token, you can choose “Tokeninfo” to view details about the selected token.

In the GTK client you can choose Tokeninfo by right clicking the token. In the Web UI you can click the button above the token view tab.

The displayed information reflect the database token table.

`TokenId` This is the internal database id of the token.

`TokenInfo` This is a field that holds many additional information like the type of the hash library used with the OATH algorithm or special information about timebased TOTP tokens.

Here you will also find information, how often a token was successfully used to authenticate and how often a user tried to authenticate with this token.

Also in this field the validity period of the token is stored. The validity period determines till when a token may be used and starting from which date a token may be used. E.g. this is used for the lost tokens (see *Lost token* (page 43)).

`OtpLen` This is the length of the OTP value. The OTP value will be truncated to this length. Also, during authentication this value is used to split the entered credential into the OTP PIN and the OTP value.

`TokenType` This is the internal type of the token. The type determines the TokenClass, that will be used to calculate the OTP value.

The list of the supported types is constantly increasing. Those types can be: hmac, totp, motp, sms, spass, radius, remote, vasco...

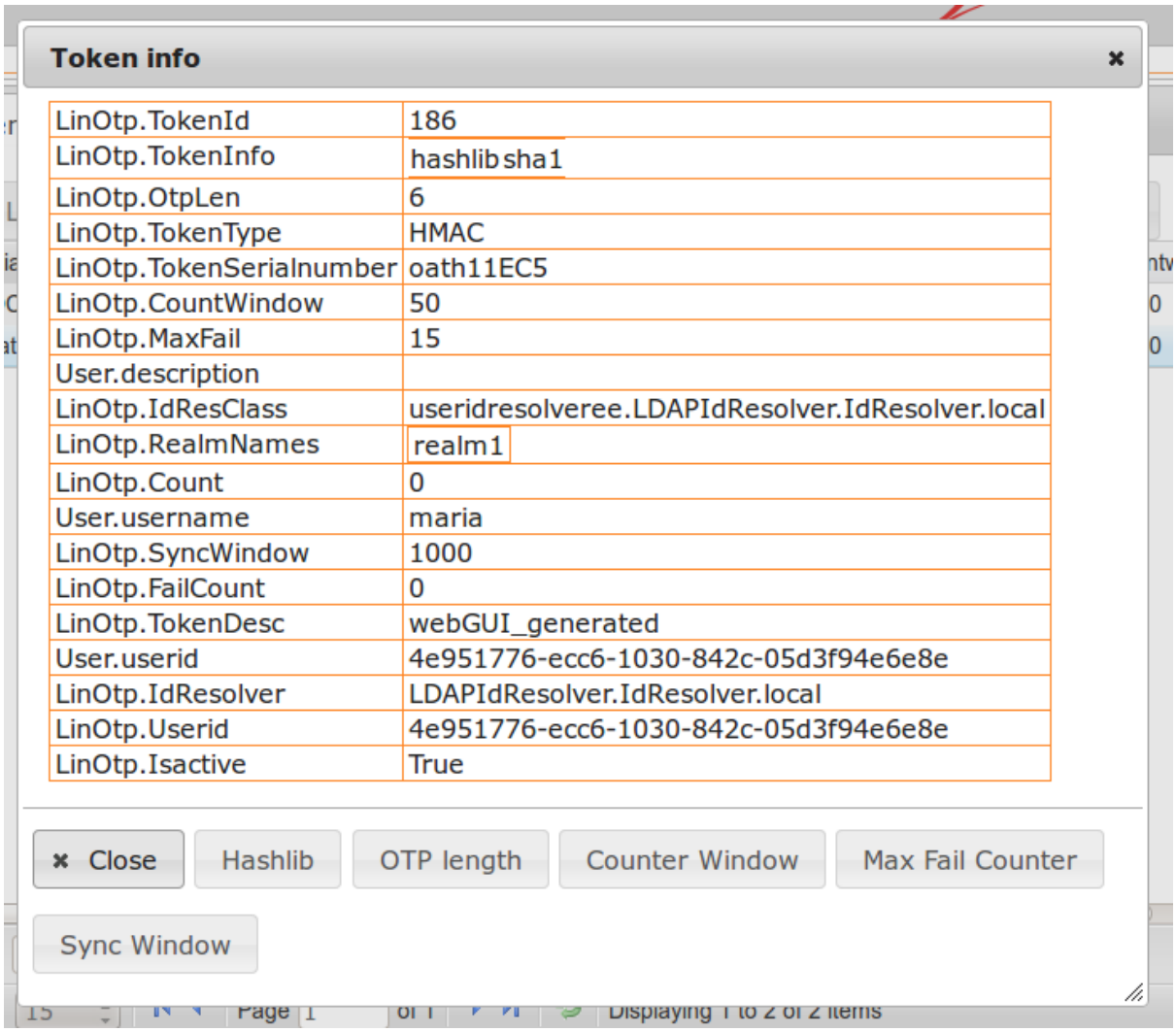


Figure 6.15: *Detailed information about the token can be viewed in the tokeninfo dialog*

Note: The type only specifies the type of algorithm to validate the OTP value or the way to distribute the OTP value. So the type "hmac" can be an SafeNet eToken PASS, a Feitian Token, a Yubikey and any other softtoken, that uses the hmac (HOTP) algorithm.

`TokenSerialnumber` This is a *unique* serial number of this very token.

The serial number also usually indicates the type of the token.

For OATH compliant tokens the first two letters specify the vendor.

Note: LSE's own tokens will always start with "LS".

The next two letters may specify the type of the token like:

- LSSP: LSE Simple Pass token
- LSRE: LSE REmote token
- LSRA: LSE RAdius token
- LSOC: LSE OCra token
- LSSM: LSE SMs token

`CountWindow` This is the window where the LinOTP server searches a matching counter to validate the OTP value.

`MaxFail` This settings corresponds with the `Failcount`. The `Failcount` gets increased, when the user fails to authenticate with this token. When the `Failcount` reaches `MaxFail` the user can not authenticate with this token anymore. (see *Reset failcounter* (page 40))

`User.description` is a variable field that gets blended in from the UserIdResolver. The contents depends on the specific UserIdResolver implementation.

`Realm names` This is the list of realms in which the token is put.

`count` This is the counter of event based tokens.

`User.username` This is the username (loginname) of the user to whom the token is assigned.

`Syncwindow` When doing a resync (either manually or automatic) the fitting counter is searched for in this window.

`Failcount` see `MaxFail`

`TokenDesc` This is the token description. Using the API it is possible to set the description to some thing of your needs. This is usually set to some kinds like "web enrolled", "temporary replacement" or if it is an SMS token to the mobile phone number.

This field has no functional purpose.

`User.userid` This is the unique identifier to find the user in the given UserIdResolver.

`IdResolver` This is the UserIdResolver, where to user, to whom the token is assigned, is located.

`Userid` This is the unique identifier to find the user in the given UserIdResolver.

`IsActive` This is the flag that indicates it the token is enabled or disabled.

6.10 Lost token

When you select one token, you can click the "lost token" button.

The lost token process can be used, when a user has lost his token. The lost token is disabled and the user gets a temporary password. The administrator or help desk personnel can tell the temporary password to the user on the phone.

To avoid that anyone can misuse this information the OTP PIN of the lost token is automatically copied to the temporary token. Thus the help desk employee will not know the OTP PIN and can not use the temporary token.

To use the lost token workflow you need to configure the corresponding policy. (see *Lost token* (page 55)). There you can define the length of the temporary password and the duration how long this temporary password is valid.

6.11 Get serial by otp

In the menu “Tools” you call the dialog “Get serial by OTP”.

Using this functionality you can enter a single OTP value to determine, what is the serial number of this token. This can be used if you find a token that has no serial number printed on it anymore. Many tokens of different vendors have a paper sticker on the back with the serial number. Sometimes the sticker falls or or the printing is rubbed of.

So this is a convenient way to identify an unknown token.

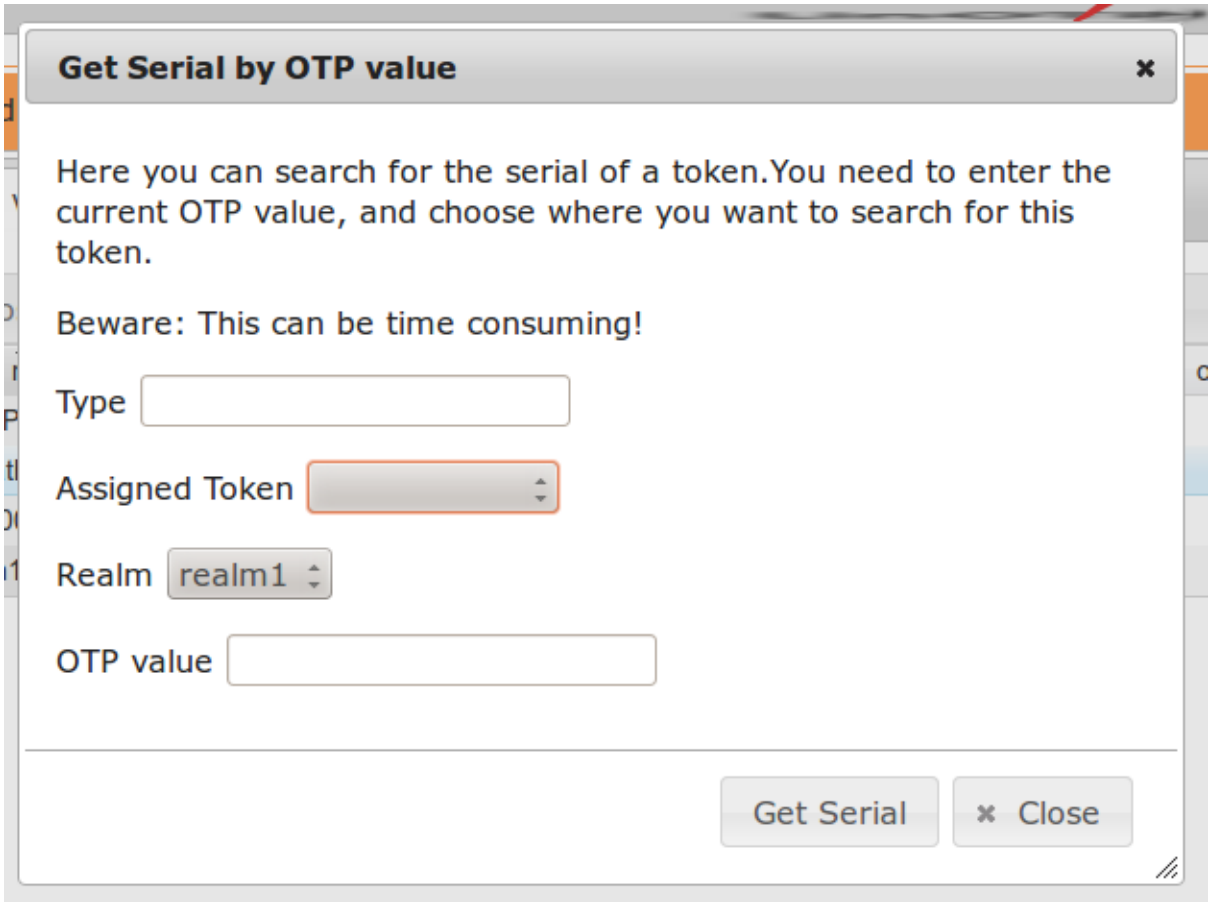


Figure 6.16: *LinOTP offers to determine the serial number by providing an OTP value*

You can limit the tokens, in which this OTP value should be searched by either defining

- the `type` of the token,
- whether the token is `assigned` or not

- and if the token should be in a certain `realm`.

Of course you need to enter the `otp value` that you want to search for.

Warning: As LinOTP needs to calculate many future OTP values of many tokens, this can be very time consuming!

6.12 Get otp

If you select only one token, you can call the function “get otp”.

This is a special function that needs several steps to get activated. How this is done is described in the policy section *Gettoken Policies* (page 59).

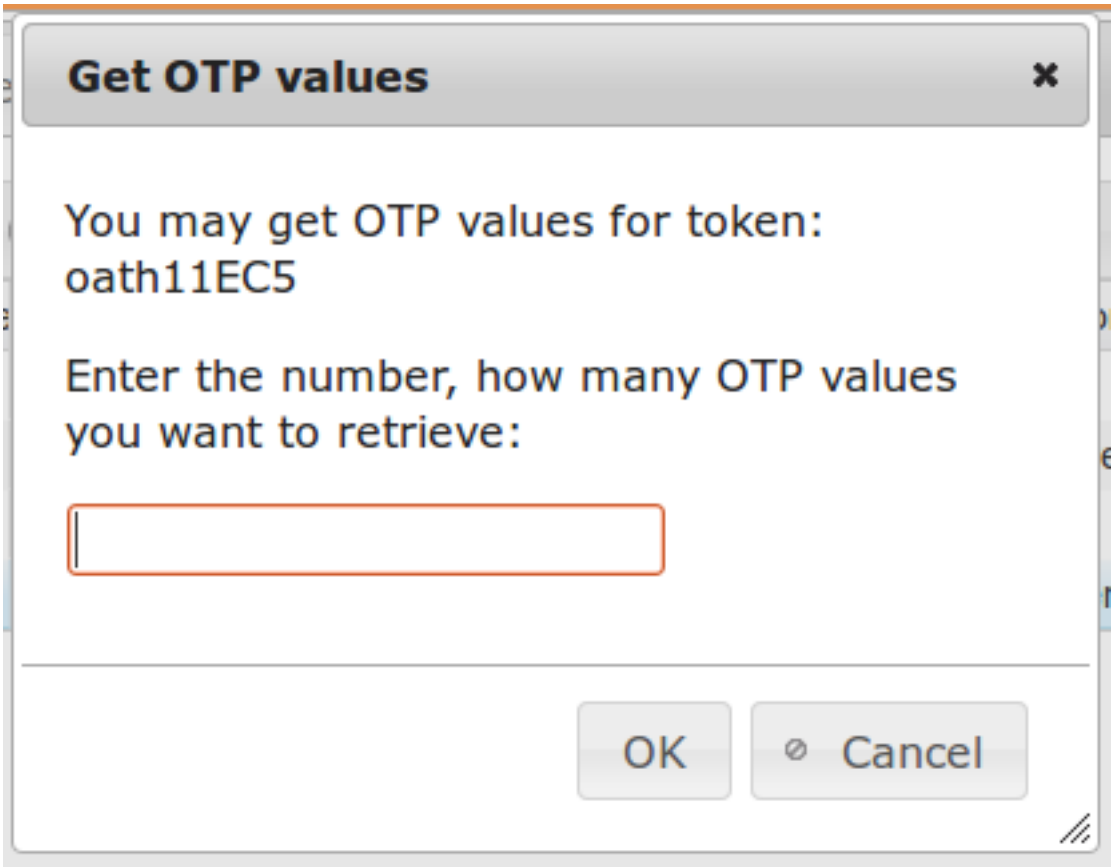


Figure 6.17: *Retrieve OTP values of a token*

Within this dialog you can enter how many of the next OTP values of the token you wish to retrieve. The maximum amount is limited by the corresponding policy.

7 | Policies

Starting with LinOTP version 2.3 also policies for users and administrators are supported. There are two different policy types. The first policy type is for users connecting to the self service portal. This way it is possible to define, which user from which realm is allowed to perform which action within the self service portal. The other policy type is for administrators managing the tokens. This way it is possible to define, that an administrator is only allowed to manage tokens within a certain realm. A policy consists of the values

`Policy name` This is the name, just to identify the policy.

`Scope` This is the scope, for which the policy is defined. Valid scopes are selfservice, admin, system, license and enrollment etc.

`Action` This is the action, which means, if this policy is valid for a certain user, the user will be allowed to do this. Several comma separated actions may be entered here.

`User`

This is the username, for whom this policy will be valid. Several comma separated username may be entered here.

For more information see *Users in policies* (page 61).

`Realm` This is the realm, for which the policy is defined.

`Client`

The client IP or subnet for which this policy is defined.

For more information see *Clients in policies* (page 62).

`Time` N/A

7.1 Selfservice policies

If you want to define policies for users accessing the selfservice portal, you need to enter “selfservice” into the scope.

You need to put a realm name into the realm field. Then this policy will work for all users within this realm logging in to the self service portal. You can also put a * into the realm field, thus the policy will be valid for all realms.

You may add clients in this policy so that you can define a different behaviour within the self service portal depending on from where the user will log in to the self service portal.

Now you may add actions, the users from this realm should be allowed to perfom. Valid actions are:

`enrollSMS` The user is allowed to self enroll an SMS token.

`enrollMOTP` The user is allowed to self enroll an MOTP token.

`webprovisionOATH` The user is allowed to enroll an HMAC oathtoken via web provisioning. [1]

[1] See this OATH token: http://code.google.com/p/oathtoken/

`webprovisionGOOGLE` The user is allowed to enroll a HMAC Google authenticator via web provisioning. [2]

`activateQR` The user is allowed to activate his QR token.

`assign` The user is allowed to assign an already imported token to himself.

`disable` The user is allowed to disable his own tokens.

`enable` The user is allowed to enable his disabled tokens.

`resync` The user is allowed to resynchronize his HMAC tokens.

`setOTPPIN` The user is allowed to reset/set the OTP PINs of his tokens.

`setMOTPPIN` The user is allowed to reset/set the MOTP PIN of his motp tokens.

`unassign` The user is allowed to unassign his own tokens.

`delete` The user is allowed to delete his own token. This token will be removed completely from the database. Nevertheless all entries for this token/serial number of course remain in the audit trail.

`getotp` The getotp tab will be displayed. Additionally the linotp.ini file needs to be adapted and a policy max_count for the token type needs to be defined.

`getserial` If this action is active, the token can be assigned by entering the OTP value. You also need to specify the action `assign` On the assign tab an additional entry for the OTP value will be displayed.

`history` If this action is active, the user will see a tab with a list (like *Audit Trail* (page 65)) of all actions he did or all the actions an administrator performed on his tokens.

Starting with LinOTP 2.5.2 you can also put users or resolvers in the *user* field. For an explaination on this take a look at *Users in policies* (page 61).

7.1.1 OTP PIN polices

Policies on how the OTP PIN should look like can be defined for the self service portal. Thus the administrator can assure, that the user chooses a OTP PIN that is secure enough to his opinion.

OTP PIN policies will be checked whenever a user tries to reset an OTP PIN in the self service portal. The policies are defined like this:

```
scope = selfservice
```

`realm` A list of comma separated realms, the policy should apply to.

`action`

Allowed actions are followed by a number or character codes. The actions can be comma separated.

- otp_pin_maxlength=12 This action would define that the user is only allowed to set OTP PIN with a maximum length of 12 characters.
- otp_pin_minlength=4 This action would define that the user needs to set an OTP PIN that is at least 4 characters long.
- otp_pin_contents=cnso This action defines, what characters need to be contained in the OTP PIN: - c: Only OTP PINs containing at least one letter character will be accepted. - n: Only OTP PINs containing at least one digit character will be accepted. - s: Only OTP PINs containing at least one special character will be accepted. - o: Only OTP PINs containing at least one special character, that is not contained in the other character groups, will be accepted.

[2] See the Google Authenticator: http://code.google.com/p/google-authenticator/

Example: So if you want your users to choose OTP PINs, that contain letters and number and that should be between 6 and 8 characters long, you should define an action like this:

otp_pin_maxlength=8, otp_pin_minlength=6, otp_pin_contents=cn

Character groups

The character groups are defined like this.

The default for characters (letters) “c” is the regular expression:

```
[a-zA-Z]
```

The default for digits “n” is the regular expression:

```
[0-9]
```

The default for special characters “s” is the regular expression:

```
[.:,;-_<>+*!/()=?$§%&#~^]
```

The character group “o” contains all characters, that are not contained in one of the above groups.

Changing the character group definition

These regular expressions can be changed in the linotp.ini file using these parameters:

```
linotpPolicy.pin_c = [a-ZA-Z]
linotpPolicy.pin_n = [0-9]
linotpPolicy.pin_s = [.:,;-_<>+*!/()=?$§%&#~\^]
```

Note: Some characters need to be escaped using the “\”.

So if you want the OTP PIN to have an upper letter at any cost, you may change the linotpPolicy.pin_c = [A-Z] and an otp_pin_contents = c. Then setting the PIN will complain, if it does not contain an upper letter.

Reversing the OTP PIN logic

The default logic is that the OTP PIN must at least contain the specified characters and may contain any more characters, digits or signs that are not required. There are two prefixes to change this logic

- -:

  The minus sign will require to contain the specified character groups, but will require, that the characters not specified are not used in the OTP PIN.

  Examples:

  - -cn: This will require the OTP PIN to contain at least a letter and a digit. But as soon as the OTP PIN contains a special character or any other character the OTP PIN will not be valid. A PIN like “a12” or “b12” would be OK, but a PIN like “abc” or “test!” would not be valid.
  - -s: The OTP PIN may only contain the specified special characters, not any digits, letters or other characters.

- +:

The plus sign will combine the specified character groups. It will require, that characters from any of the character groups are contained in the PIN, but characters not in the character groups must not be contained in the PIN.

Example:

- +cn: The PIN needs to contain letters or digits, but must not contain any special characters. A PIN like “123”, “abc”, “a12” would be OK.

Example: If you want to have the user always use PINs that are exactly 4 digits long, you can set:

otp_pin_maxlength=4, opt_pin_minlength=4, otp_pin_contents=-n

7.1.2 Retrieving OTP values

In the scope=selfservice you can define the following actions:

- max_count_dpw=<int> for retrieving OTP values of Tagespasswort tokens.
- max_count_hotp=<int> for retrieving OTP values of HOTP/HMAC tokens.
- max_count_totp=<int> for retrieving OTP values of TOTP tokens.

These actions are the equivalent to the gettoken actions (see *Gettoken Policies* (page 59)). When these actions are defined, the user in such a realm is allowed to retrieve future OTP values of his own OTP tokens. You need to also specify the policy scope=selfservice, action=getotp and edit the linotp.ini file to enable OTP retrieval.

7.2 Admin Policies

If you want to define policies for administrators managing tokens, you need to enter “admin” at the scope field.

Note: As long as there is no policy in the scope “admin” defined, all admins will have access to all functionality. This is the old behavior of LinOTP version 2.2. As soon as the is there first admin policy defined, admin policies are checked for each and every administrator.

You may enter several comma separated administrator names into the user field.

You may enter several comma separated actions into the action field. Entering an asterisk ‘*’ here, will allow the administrators to perform all actions.

You may enter several comma separated realm names into the realm field. Entering an asterisk ‘*’ here, will allow the administrators to perform these actions in all realms.

Valid actions are:

`initSPASS` The administrator is allowed to enroll Simple Pass tokens.

`initHMAC` The administrator is allowed to enroll HMAC tokens.

`initETNG` The administrator is allowed to enroll an eToken NG-OTP. As the eToken NG-OTP is also an HMAC token, the administrator will also need to have the action initHMAC.

`initSMS` The administrator is allowed to enroll SMS tokens.

`initMOTP` The administrator is allowed to enroll mOTP tokens.

`initREMOTE` The administrator is allowed to enroll remote tokens.

`initRADIUS` The administrator is allowed to enroll RADIUS tokens.

`enable` The administrator is allowed to enable tokens.

`disable` The administrator is allowed to disable tokens.

`set` The administrator is allowed to set token properties like MaxFailCount, SynWindowSize.

`setOTPPIN` The administrator is allowed to set the OTP PIN of tokens.

`setMOTPPIN` The administrator is allowed to set the mOTP PIN of mOTP tokens.

`setSCPIN` The administrator is allowed to set smartcard PINs (SO and User) in the database.

`resync` The administrator is allowed to resynchronize HMAC tokens.

`reset` The administrator is allowed to reset the Failcounter of tokens.

`assign` The administrator is allowed to assign tokens to users.

`unassign` The administrator is allowed to remove the assignment from tokens to users.

`import` The administrator is allowed to import token description XML files.

`remove` The administrator is allowed to delete tokens from the database. These tokens are completely removed from the system.

`userlist` The administrator is allowed to view the users in this realm.

`manageToken` This action allows to add and remove tokens from a realm. To be able to move tokens from a Realm A to a Realm B, the administrator needs the action manageToken both in Realm A and in Realm B.

`getserial` This action allows the administrator to use the getSerialByOtp tool.

`copytokenpin` This action allows the administrator to use the copyTokenPin tool. This is used to copy the OTP PIN of one token to another token without the administrator knowing the PIN.

`copytokenuser` This action allows the administrator to use the copyTokenUser tool. This is used to copy the User of one token to another token.

`losttoken` This action allows the administrator to use to lost token function.

`getotp` This action allows the administrator to run the get OTP workflow for tokens in the specified realm.

7.3 System policies

Using policies in the scope “system”, it is possible to define, which administrator is allowed to set system configurations. As the system configuration also contains credentials of the UserIdResolvers, restricting the read access to the system configuration can make sense. The following policy grants read access to the system configuration:

- scope = system
- action = read
- user = <comma separated list of administrator names>

The following policy grants write access to the system configuration:

- scope = system
- action = write
- user = <comma separated list of administrator names>

System policies do not refer to any realm.

Note: If no policy with scope “system” is defined, all administrators get full access to the system configuration according to the Apache2 configuration file.

If a realm admin has no read access to the scope system, he can only retrieve the system configurations that are vital for his administrative tasks. E.g. when he tries to retrieve the list of the Realms, he will

only see the realms he as certain rights in. This way you can avoid that a realm admin can see what other realms exist.

7.4 License policies

The license policy defines, which administrator is allowed to set the license of the LinOTP 2 server. To grant the right of setting the license, the following policy needs to be defined:

- scope = license
- action = setlicense
- user = <comma separated list of administrator names>

The license policy does not refer to any realm.

Note: If no policy with scope "license" is defined, all administrators are allowed to set the license according to the Apache2 configuration file.

7.5 Enrollment Policies

Enrollment Policies are checked during the enrollment process of a token. Be it during administrative enrollment or user self enrollment.

7.5.1 Token limits per Realm

You may limit the number of tokens that are managed within a realm. This is done by defining a policy consisting of

- Policy name: This is the unique name of the policy.
- Scope: You need to set this to **enrollment**.
- Realm: Enter the name of the realm here.
- Action: This field is comma separated. It should contain the keyword **tokencount = <tokennumber>**.

Using the action tokencount you may allocate specified numbers of tokens to your different realms.

You may for example have a license with 1000 tokens, but one of your five realms should only be able to manage a maximum of 80 tokens, since this realm is managed by another department.

Note: Each realm, for which no tokencount is defined, may use as many tokens as possible.

7.5.2 Token limits per user

You may limit the number of tokens that can be assigned to a user. This is done by defining a policy consisting of

- Policy name: This is the unique name of the policy.
- Scope: You need to set this to **enrollment**.
- Realm: Enter the name of the realm, that contains the users who should be restricted in the number of tokens owned.

- Action: This field is comma separated. It should contain the keyword **maxtoken = <tokennumber>**.
- User: Here you can enter a list of usernames or resolvers. See *Users in policies* (page 61) for more details.

Note: If there is no maxtoken defined for a realm, then you may enroll as many tokens as possible for a single user.

7.5.3 Random OTP PIN

During enrollment a random OTP PIN may be set. This is done per realm by defining the policy:

- Policy name: This is the unique name of the policy.
- Scope: You need to set this to **enrollment**.
- Realm: Enter the name of the realm, that contains the users who should be restricted in the number of tokens owned.
- Action: This field is comma separated. It should contain the keyword **otp_pin_random = <length>**. This will create a random OTP PIN during token initialization of the given length.
- User: Here you can enter a list of usernames or resolvers. See *Users in policies* (page 61) for more details.

Note: If there is no otp_pin_random defined for a realm, than no PIN is set during enrollment. If a random OTP PIN is set, it is not possible to view this PIN. This means, that the user or the administrator has to reset the OTP PIN.

7.5.4 Encrypted OTP PIN

During setting an OTP PIN this policy defines, if the OTP PIN will be stored as hashed value or as an encrypted value. The default behavior is to store the OTP PIN as a hashed value.

- Policy name: This is the unique name of the policy.
- Scope: You need to set this to enrollment.
- Realm: Enter the name of the realm, that contains the tokens, the policy should be valid for.
- Action: This field is a comma separated list. It should contain the keyword otp_pin_encrypt = 1.

Note: The encrypted value can be decrypted again. This is used in case of getOtp function. Usually it is a good idea to leave this untouched.

7.5.5 Tokenlabels

It is possible to set a token label for the Google Authenticator and OATH token. You may define a policy, what this token label should be made of.

- Policy name: This is the unique name of the policy.
- Scope: **enrollment**
- Realm: Enter the name of the realm, that contains the tokens, the policy should be valid for. The tokens of users in this realm will be labeled accordingly.

- Action: This is a comma separated list. It should contain **tokenlabel=<string>**, where <string> is a regular expression, that works like this: - <u> will be replaced with the name of the user - <r> will be replaced with the name of the realm - <s> will be replaced with the serial number of the token.
- User: Here you can enter a list of usernames or resolvers. See *Users in policies* (page 61) for more details.

If no policy can be found or the token is not assigned to any user, then the token will be labeled with the serial number.

Example: tokenlabel=<u>@<r> will result in a label like username@realm1 (username@realm1). tokenlabel=ourtokens_<s> will result in something like outtokens_oath1278489

7.5.6 Auto Assignment

Enrolling tokens to users can be a hard process to do. LinOTP provides a way to assign existing tokens to users in the most easy way. If Auto Assigment is configured for a realm tokens can be assigned like this:

First you need to import the token seed file and set the token realm of these tokens.

Then you need to configure a autoassignment policy.

After having done so, the enrollment workflow is as follows:

1. Tokens are set to realmA but not assigned.
2. The user, who is in realmA, grabs any of those realmA tokens.
3. Now he authenticates against LinOTP using his userstore password, which will be either the passwd-password, the LDAP-password or his SQL-password and an OTP value from the token he just grabbed.
4. LinOTP will check,
    - if the autoassignment policy is set for this realm,
    - if the user has not yet any token assigned,
    - if the user provided the right password,
    - and then look up the token for the given OTP value in the tokens of realmA that are not yet assigned.
5. LinOTP then will assign the found token to the user and set the userstore password as OTP PIN.
6. The user is successfully authenticated and will have to use this very token for all future logins.

The autoassigment policy is defined like this:

- Policy name: This is the unique name of the policy.
- Scope: **enrollment**
- Realm: Enter the name of the realm, that contains the tokens, the policy should be valid for. The tokens of users in this realm will be labeled accordingly.
- Action: This is a comma separated list. It should contain **autoassignment=<int>**, where <int> is either 6 or 8, depending on the length of the tokens OTP value.
- User: Here you can enter a list of usernames or resolvers. See *Users in policies* (page 61) for more details.

7.5.7 Lost token

When a user looses a token, the help desk can enroll a temporary token that will replace the lost token. The lost token will be disabled. The OTP PIN of the lost token will be used for the new temporary token. The administrator or the help desk employee do not need to know and can not know the OTP PIN. The temporary token will be a long static password and will only be valid for a restricted amount of days. The length of the password and the validity period can be configured using an enrollment policy:

- Scope: **enrollment**
- Action: **lostTokenPWLen=<length of password>** (default = 10)
- Action: **lostTokenValid=<length of validity period in days>** (default = 10)
- Action: **lostTokenPWContents=<contents of the temporary password>**
- User: Here you can enter a list of usernames or resolvers. See *Users in policies* (page 61) for more details.

Contents

The contents is described by these characters:

C: The password will contain upper characters.

c: The password will contain lower characters.

n: The password will contain digits/numbers.

s: The password will contain special characters.

Example: A temporary token with lostTokenPWLen=10, lostTokenValid=30 will get a 10 characters static password created and will be valid for the next 30 days.

7.6 Authorization Policies

Starting with LinOTP 2.5 Authorization policies are supported. This means you can define, which users from which realm are allowed to login to which client. Therefore LinOTP grabs the HTTP client, that is doing the HTTP request. In case of using a RADIUS-Server, it will always be the IP address of the RADIUS server, that talks to the LinOTP server. Therefore the RADIUS server can impersonate the client by sending an additional URL attribute “client” containing the RADIUS clients IP address to the LinOTP server.

In the system config you may define “Override Authentication client” which is a comma separated list of IP addresses, that are allowed to send other client IPs.

7.6.1 Authorization

To define an authorization policy you need to do the following:

- Policy name: This is the unique name of the policy.
- Scope: This needs to be **authorization**.
- Action: This needs to be **authorize**. There are no other actions at the moment.
- Realm: This is a list of realms or ‘*’ for all realms. A user must be member of this realm to be authorized to authenticate to the defined clients.
- User: This is a list of users or resolvers. See *Users in policies* (page 61) for more details.
- Client: This is a list of IP addresses or IP subnets.

Such a policy will define, that the users from this realm are allowed to login on the specified client. If a policy for users is specified, these users are not allowed to login at other clients than defined in this policy.

Also, other users, that are not mentioned in this policy, are not allowed to login to the clients in this policy.

Note: If a policy for a client 192.168.20.10 and a UserA in Realm1 is specified, then the UserA *and* the UserB in Realm1 will not be allowed to login to client 192.168.20.11.

7.6.2 Token Types

Using this policy you may define, that a user will only be allowed to login with the correct token type. You may add several token types to a policy, so that the user may choose between two or three token types.

- Policy name: This is a unique name of the policy.
- Scope: You need to set this to **authorization**.
- Realm: Enter the name of the realm. These users will get a customized text.
- Action: Use the format **tokentype=HMAC TOTP PW**. (may contain '*').
- User: This is a list of users or resolvers. See *Users in policies* (page 61) for more details.
- Client: This is a list of IP addresses or IP subnets.

Note: The entered password gets validate, i.e. if the user enters a correct OTP PIN and OTP value this is validated and the counter is increased, so that those values can not be used to login to another client.

If the user enters valid OTP values, the Fail Counter is not increased, even if he is not allowed to login with this token type on this very client.

Note: The user specified in this policy may only login at the specified client with the specified token. I.e. this specified user may login with another token type at another client. This also means that a user *not* specified in this policy can also login to this specified client with another token type.

So if you want make sure, that at a specified client only one token type is allowed to login, you must make sure that all users are either contained in this policy or that users, who should not be allowed to login at that client are defined in an **authorize** policy.

7.6.3 Setting Realms

The *setrealm* policy can be used to define a fixed realm for an authenticating client. When this policy is triggered the realm of the authenticating user is overwritten to the realm, that is defined in the policy.

- Policy name: This is a unique name of the policy.
- Scope: You need to set this to **authorization**.
- Realm: Enter the name of the realm for which this policy should be valid.
- Action: Use the format **setrealm=new_realm** to define, what should be the new realm of the authenticating user.
- User: This is a list of users. You can not define resolvers here, since the user in might not be contained in the original realm! See *Users in policies* (page 61) for more details.
- Client: This is a list of IP addresses or IP subnets.

Note: Using this policy you can e.g. rewrite one realm to another realm on one client. Or you can define clients, where only users from a single realm, which is not the default realm, may authenticate without having to provide the realm name.

7.7 Authentication Policies

Some authentication policies are also making use of the client field. This is described in the policy.

7.7.1 OTP PIN variants

You may define the way of authenticating the users in the realms.

Define a policy like this:

- Policy name: This is the unique name of the policy.
- Scope: You need to set this to authentication.
- Realm: Enter the name of the realm here.
- Action: This field is comma separated. It should contain the keyword **otppin** = <id>.

  <id> can have one of the following values:

  - 0: This is the default behavior. In addition to the OTP value the user needs to enter a fixed OTP PIN, that is defined per token.
  - 1: In addition to the OTP value the user must enter the password from his user database, e.g. his LDAP or Active Directory password (in case of LDAPUserIdResolver) or his password in the Passwd-File or in the SQL Database.
  - 2: The user does not need to enter any additional OTP PIN. He will authenticate only with the OTP value.
- User: This is a list of users or resolvers, for whom this policy will be valid. Please see *Users in policies* (page 61).
- Client: This is a list of IP addresses or IP subnets.

Note: Using this policy you can setup scenarios, where a user needs to provide a PIN, when she logs in from the SSL VPN but does not need to provide an OTP PIN when logging in from a Radius Credential Provider.

7.7.2 Automatic SMS sending

Using this policy you may define, that a new SMS is sent after the user has successfully authenticated with an SMS OTP. This way, the user does not need to do some additional step for triggering the sending of a new SMS. When you use the client parameter, you can define, that an automatic SMS will be triggered when the user logged in to the Terminal Server but not to the SSL VPN.

- Policy name: This is the unique name of the policy.
- Scope: You need to set this to authentication.
- Realm: Enter the name of the realm here.
- Action: This field is comma separated. It should contain the keyword **autosms**. This keyword works as a boolean and takes no parameter.
- User: This is a list of users or '*' for all users, for whom this policy will be valid.

- Client: This is a list of IP addresses or IP subnets.

7.7.3 SMS Text

Using this policy you may define the text content of the SMS. Usually the SMS only contains the OTP value. But you may change this per realm.

- Policy name: This is a unique name of the policy.
- Scope: You need to set this to **authentication**.
- Realm: Enter the name of the realm. These users will get a customized text.
- Action: Use the format **smstext=** ”Your OTP <otp> for your token <serial>”. You may use text, that can be send via SMS. The place holders <otp> and <serial> will be replaced by the OTP value and the serial number of the token. You do not need to add the serial number!

7.7.4 Authentication Passthrough

When this is activated, users who have no token assigned, will be authenticated against the password in their UserIdResolver.

- Policy name: this is a unique name of the policy.
- Scope: You need to set this to **authentication**.
- Realm: Enter the name of the realm.
- Action: **passthru**
- User: This is a comma seperated list of usernames or resolver names. Please see *Users in policies* (page 61).
- Client: This is a list of IP addresses or subnets this policy is valid for.

Note: Using this policy you can setup smooth rollout scenarios. Every user will have to authenticate with her active directory password until the user gets an OTP token enrolled. In such an enrollment scenario you could combine this policy with the OTP PIN policy `otppin=1`. Thus a user will authenticate with the AD password and will have to **add** the OTP value to the fixed AD password as soon as she has a token assigned.

7.7.5 URL for QR-TAN token

The LSE QR-TAN token has a half automatic mode, where the App tries to send the TAN to a configured URL.

This URL is transferred to the App during the enrollment process. You can define different URLs for each realm.

To do so you can set up a policy like this:

- Policy name: This is a unique name of the policy.
- Scope: **authentication**
- Realm: One realm, a comma separated list of realms or ‘*’
- Action: **qrtanurl=** <URL> The URL should be something like `https://yourserver/ocra/check_t`

7.8 Gettoken Policies

LinOTP allows you to retrieve current or future OTP values of tokens for certain scenarios. This can be used for Tagespasswort-Tokens, if the application does not support RADIUS or any other authentication protocol or for the scenario where offline OTP authentication is required. To define such a gettoken policy do the following:

- Policy name: the unique name of the policy
- Scope: You need to set this to gettoken.
- Action: can be something of max_count_dpw=<int>, max_count_hotp=<int>, max_count_totp=<int>
- User: The administrative account that is allowed to retrieve OTP values.
- Realm: The name of the token realm the policy is valid for. This can be a list or a '*'.

Note: In contrast to other policies retrieving of OTP values will not be allowed unless a policy is defined.

If you want to run the gettoken/getotp function from the Management WebUI, you also need to set the admin policy (action getotp) for the specific realm. (see section *Admin Policies* (page 50))

As retrieving OTP values must be seen as a kind of security breach, you need to do more configuration to activate the gettoken functionality. For configuring this see section *Retrieving OTP values* (page 77).

OTP values may be retrieved using the API calls `/gettoken/getotp` and `/gettoken/getmultiotp`. The policy will define which kind of tokens would be allowed to have their OTP values retrieved by either defining max_count_dpw for Tagespasswort-Tokens, max_count_hotp for HMAC tokens or max_count_totp for TOTP tokens. These action parameters require an integer parameter, which specifies how many OTP values will be returned at max. So if 100 OTP values would be requested but max_count_hotp=50 only 50 values will be returned. LinOTP will only return OTP values if the tokens realm is contained in the policy.

7.9 Audit Policies

You may define, which administrator is allowed to view the audit trail.

Define a policy like this:

- Policy name: This is the unique name of the policy.
- Scope: You need to set this to audit.
- Action = view.

Note: If no audit policy is defined every administrator is allowed to view the audit trail.

7.10 OCRA Policies

You may define, which administrator or system account is allowed to acces the ocra methods.

Define a policy like this:

- Policy name: This is the unique name of the policy.
- Scope: You need to set this to **ocra**.
- **Action: This is something of:**

- **request** to access *ocra/request*
- **status** to access *ocra/status*
- **activationcode** to access *ocra/activationcode*
- **calcOTP** to access *ocra/calculateOtp*.

Note: If no ocra policy is defined, no administrator or system account is allowed to access the above methods!

7.11 Policy checker

When configuring complicated policies, you may wonder if you configured the policies correctly and if the user `maria` in realm `realm1` is able to enroll a Google Authenticator in the selfservice portal.

For this you can use the policy checker in the Web UI that can be accessed via *Tools* -> *Check policy*. (see *The policy checker dialog* (page 60)).



Figure 7.1: **The policy checker dialog**

The policy checker simulates the action with the user, that you enter in the dialog.

Then the system tells you, which policy will be applied. Therfore you always need to enter the username, the realm and an action, that you want to check.

Note: You can also check enrollment or authentication policies like `maxtoken` or `otppin` to determine, how many tokens a certain user is allowed to own of if a certain user needs to authenticate with his OTP PIN or his directory password.

7.12 Importing and exporting policies

You can export and import the policies. This can be done from the policy tab in the WebUI or from the policy dialog in the GTK client.

Note: When exporting policies always *all* policies are exported.

The policies are written to a file in a config file format. Each section of a this export file is the name of the policy. The key/value-pairs in such a section are the settings of this policy.

You can edit the policy file in your preferred editor and import the policy again. Exporting and importing policies is helpful to backup policies or to transfer policies from one system to another or the prepare the policy definitions in a text file and import these ready made policies to your LinOTP system.

Note: When importing policies existing policies are not flushed. The imported policies are added to the already existing policies. If a policy with the same name already exist, then this policy gets overwritten by the imported policy.

Warning: The WebUI imports the policy file as a whole while the GTK client imports one policy after another. So if a first policy defines that your admin user has no write access to the system config, importing the later policies will fail! This only happens with the GTK client!

See *Best practice - policy example* (page 63) for policy examples.

7.13 Users in policies

The user field in the policies need a deeper explanation.

Within the scope *admin*, *license*, *audit* and *system* the user field is used to enter a list of administrator names.

In the scope *selfservice*, *authorization*, *enrollment* and *authentication* the user field can be used to specify users and resolvers. Thus the user field can be used to split policies within realms by users or resolvers.

Note: When using the user field you should always define a certain realmname in the realm field and not use the * for the realms.

Users are specified with the normal username like:

```
user1, user2, user3
```

Resolvers are specified with the name of the resolvers followed by a colon, to mark this as a resolver:

```
resolvername1:, resolvername2:
```

Note: It is also possible to mix users and resolvers like this:

user1, resolvername1:, user2, user3

The policy logic works like this:

1. If a policy in the scope is found for this very user, than this policy is applied and all other policies in this scope not containig the user are ignored.
2. The resolver of the user is determined. If a policy in the scope for this resolver is found, this policy is applied and the other policies are ignored for this user.

3. If within this scope no policy with the very username and no policy with a resolver the user is in is found, than the policies containing no user entries are applied for this user.

Imagine a realm *realm1* with two resolvers *resolv1* and *resolv2*. There are four users, *user1a*, *user1b* and *user1c* in *resolv1* and *user2* in *resolv2*.

The following policies are defined:

```
[pol1]
scope = selfservice
action = webprovisionGOOGLE
realm = realm1
user = ""

[pol2]
scope = selfservice
action = webprovisionGOOGLE, setOTPPIN
realm = realm1
user = user1a

[pol3]
scope = selfservice
action = webprovisionGOOGLE, setOTPPIN, disable
realm = realm1
user = user1b, resolv2:
```

pol1 will be applied for *user1c* as this policy does not define any user and as *user1c* or his resolver does not appear in any other policy.

pol2 will be applied for *user1a* since this user is explicitly listed in the policy.

pol3 will be applied for *user1b* as the user is listed in the policy and it will be applied for *user2* since the resolver of this user is listed in the policy.

7.14 Clients in policies

The client field in the policies can include or exclude clients.

In the scope *selfservice*, *authorization*, *enrollment* and *authentication* the client field can be used to specify for which client the policy is valid. The IP address of the client, from which the request originated, is determined and checked against the policies.

You can enter a list of IP addresses or networks in CIDR [3] notation seperated by commas:

```
172.16.15.0/16, 10.0.0.0/8, 192.168.1.1
```

When combining several policies it can be necessary to exclude client IPs from a policy. You can do this by using the prefix `-` or `!`:

```
172.16.15.0/16, -172.16.15.123, 10.0.0.0/8, -10.0.2.0/16, 192.168.1.1
```

Such a policy definition:

```
[special_user]
realm = realm1
client = 172.16.200.147
user = administrator
time = ""
action = "authorize, "
scope = authorization
[all_users]
realm = realm1
```

[3] Classless Inter-Domain Routing

```
client = "172.16.200.0/24, -172.16.200.147"
user = *
time = ""
action = "authorize, "
scope = authorization
```

would mean, that only the `administrator` is allowed to authenticate on all computers on the `172.16.200.0/24` network. All other users can authenticate on the network except at the machine `172.16.200.147`.

7.15 Best practice - policy example

Here is an example to give you an idea about the policies.

Think about a scenario, where there is an administrator “superadmin”, who is allowed to do anything within LinOTP. Then there is an administrator “enroller”, who is allowed to enroll and manage tokens. Finally there is the user help desk “uhd”, who is allowed to solve problems of the users.

The users are allowed to enroll Google Authenticators in the selfservice portal.

Now lets take a look at the policies.

7.15.1 Administrators

The policies for the “superadmin” look like this:

```
[policy1]
scope = admin
action = *
user = superadmin
realm = *

[license1]
scope = license
user = superadmin
action = setlicense
```

According to policy `policy1` and `license1` the “superadmin” is allowed to do all administrative tasks in all realms and also to set the license.

There need to be policies for the other administrators:

```
[policy2]
scope = admin
action = initHMAC, resync, setOTPPIN, userlist, assign, unassign, remove, reset, set, import
user = enroller
realm = *

[policy3]
scope = admin
action = reset, setOTPPIN, losttoken, resync, enable, disable
user = uhd
realm=*
```

The administrator “enroller” is allowed to initialize tokens, set PINs, assign and import tokens. So roughly all token managing tasks. He is not allowed to enroll any other tokens than HMAC tokens and he is not allowed to move tokens between realms (missing action `manageToken`).

According to policy `policy3` the administrator “uhd” is only allowed to reset the failcounter of a token, to set the PIN or enable or disable the token. The user help desk is not allowd to enroll new tokens or remove tokens.

7.15.2 System

To avoid the changing of policies and also other system settings, we need to define some `system` policies:

```
[syspol1]
scope = system
action = read, write
user = superadmin
```

Only the “superadmin” is allowed to read the system configuration and also change the system configuration. All other administrators can not see or change the system config. i.e. they also can not see the policies.

7.15.3 Selfservice

It is now completely up to you which kind of selfservice policies you want to define. But given, that the users should be able to enroll a Google Authenticator on their own, this could look like this:

```
[selfpol1]
scope = selfservice
realm = realm1
action = webprovisionGOOGLE, reset, resync, setOTPPin, disable

[enroll1]
scope = enrollment
realm = realm1
action = maxtroken = 2
```

The policy `selfpol1` defines, that the user is allowed to enroll a Google Authenticator and that he can set the PIN and reset the failcounter in the selfservice portal. He would be also able to `disable` his token - which could only be enabled by the user help desk again. The policy `enroll1` limits the allowed number of tokens of a user to a maximum of 2.

8 Audit Trail

Starting with LinOTP 2.4 you may also write an Audit Trail. The Audit Trail contains digitally signed log messages, that shall inform you, which user or administrator did what with which token.

For all events a consecutive number, the timestamp, the LinOTP server name and a digital signature is logged. Depending on module, other information are logged, too.

8.1 Logging Scopes

8.1.1 Selfservice portal

The fact that a user logs in to the audit trail is logged as action “selfservice/index”. All actions a user may perform within the selfservice portal are also written to the Audit Trail.

The following information is logged:

- action: can be:
  - userassign
  - userdisable
  - userinit
  - userresync
  - usersetmpin
  - usersetpin
  - userwebprovision
- user: the user logon name.
- realm: the realm of the user.
- serial: The token serial number, on which the user preformed the action.
- token_type: The type of the token, on which the user preformed the action.
- success: whether the action was successful or not.
- info: additional information in case of failure.

8.1.2 Validate

If a user is authenticating the following actions and information are logged:

- action: This can be
  - check
  - simplecheck

  - check_s
- user
- realm
- serial
- token_type
- success
- info

8.1.3 Audit Trail

When an administrator is viewing the Audit Trail, this event is also logged with the following information:

- action: audit/search
- administrator: The name of the administrator
- info: Contains the additional search pattern

8.1.4 Administrative Tasks

All administrative Tasks are logged with these additional information:

- action: This can be one of the many admin/<tasks> actions.
- action-details: in case of the action admin/set, there are additional information, what feature of the token was set.
- administrator: The name of the LinOTP administrator performing this task.
- user: The name of the user on whose token this task was performed.
- serial: The token serial number on which the task was performed.
- token_type: The type of the token on which the task was performed.
- success
- info: more information in case of a failure or new values that were set, e.g. the new value of the OTP length when the OTP length was set.

8.1.5 System configuration

All system configuration tasks including policy settings, UserIdResolver and realm definitions are logged with the following information:

- action: The name of the action.
- action-detail: If a key is set, this contains the name of the key.
- administrator: The name of the administrator performing the task.
- success
- info: more information in case of failure or the new value if a key was set.

8.1.6 Setting License

When an administrator is setting the license, the following information are logged:

- action: license/setLicense
- administrator
- success
- info: contains either additional error messages or in case of success information about the newly set license.

8.2 Viewing the Audit Trail

You may view the Audit Trail in the tab “Audit Trail”.

8.3 Configuration

The configuration of the Audit Trail can only be done via the linotp.ini file. This is a security feature to avoid LinOTP admins to configure the Audit Trail. The Audit Trail is written by a module, that you can choose or reprogram. LinOTP comes with two modules:

- File Audit Trail
- SQL Audit Tail.

Configure the File Audit Trail like this:

```
linotpAudit.type = linotpee.lib.Audit.FileAudit
linotpAudit.file.filename = /var/log/linotp/audit.log
linotpAudit.file.audit_num = %(here)s/audit_num
```

Configure the SQL Audit Trail like this:

```
linotpAudit.type = linotpee.lib.Audit.SQLAudit
linotpAudit.sql.url = mysql://<user>:<password>@sqlserver/<Database>
```

If the user got the right to create database tables, the table will also be automatically created on first use.

The Audit log entries get digitally signed so that they can not be manipulated. You can generate a public/private key pair like this:

```
openssl genrsa -out private.pem 2048
openssl rsa -in private.pem -pubout -out public.pem
```

Now you can specify the location of the private and public key for the Audit-Modules in the `/etc/linotp2/linotp.ini` file:

```
linotpAudit.key.private = /etc/linotp2/private.pem
linotpAudit.key.public = /etc/linotp2/public.pem
```

You should reduce the access right of the private key for the owner linotp to 400.

Note: You can create the signing keys using the tool `linotp-create-auditkeys -f <linotp.ini file>`.

The default name of the database audit table is “audit”. On certain databases like Oracle this is a reserved keyword, therefore you can define a prefix to ensure that there is no name collision:

```
linotpAudit.sql.table_prefix = linotp_
```

will make LinOTP trying to write the audit data to the table “linotp_audit”.

8.4 Searching the Audit Trail

You can also search with the different columns.

The search depends on your Audit Module. The File Audit Module allows common regular expressions. The SQL Audit Module accepts searching like SQL WHERE clauses

8.4.1 Basic search in File Audit

When searching in columns regular expressions are allowed.

If you want to find all tokens with a serial number that begins with F789, you may search for:

```
F789.*
```

If you want to find all events on a certain date between 12 am and 2 pm, you search for:

```
2001-05-27 1[2-4]:.*
```

8.4.2 Extended Search in File Audit

Using the “extended search” you may find more complex scenarios, e.g. you may find all events, when the user “schmidt” tried to authenticate on May 27th without success.

Choose “extended search” and concatenate the patterns with a “;”:

```
date=2011-05-27;action=validate/.*;user=schmidt;success=False
```

If you have a search pattern with conditions where any one may fit, you may change the AND-concatenation to an OR-concatenation by added the keyword:

```
or = true
```

8.4.3 Search in SQL Audit

You can also do basic search and extended search with the SQL Audit Module. But the search pattern will only accept the % character like this:

```
date=% 11:%; action=system%
```

Which will give you all system configuration events at 11 am.

For productive use it is highly recommended to use the SQL Audit, as the File Audit gets really slow and unusable with a high audit volume.

9 SMSProvider for SMS OTP Tokens / Mobile TANs

Starting with version 2.2 LinOTP also supports sending one time passwords via SMS. In this case an SMS OTP token needs to be created. Internally this token is a HMAC token. The token object gets a mobile phone number and this token is assigned to the user.

9.1 Overview

9.1.1 Triggering SMS

When calling the API `/validate/smspin` the user is authenticated by the username and OTP PIN. The sending of an SMS with a OTP value is triggered. This OTP value is valid for a configurable amount of time. A sample webpage can be used at

https://your-server-ip/auth/requestsms.

Another possibility of triggering SMS is when the user authenticates with username and PIN. When username and PIN are passed to the authentication URL like `/validate/check` or `/validate/simplecheck`, the user is rejected, but, if the PIN is correct, the sending of an SMS is triggered.

Note: Thus SMS can be easily used in conjunction with the RADIUS protocol. The user first needs to authenticate at the RADIUS client with username and PIN. The user will be rejected. After that an SMS will be sent to the user. Then the user needs to authenticate with username and PIN+OTP (from the SMS) and the user will be successfully authenticated.

9.1.2 Configuration

These three values can be configured in the server configuration:

- SMSProvider
- SMSProviderConfig
- SMSProviderTimeout

9.2 SMSProvider

An SMSProvider is a LinOTP module, that defines how SMS are sent. At the moment LinOTP comes with two different kinds of SMSProviders.

9.2.1 HttpSMSProvider

In this case LinOTP uses public available SMS gateways. There are providers like clickatell.com (http://clickatell.com) or smsout.de (http://smsout.de) that can create an SMS by calling a certain well formed URL. The LinOTP HttpSMSProvider module is designed very flexible, so that every SMS Gateway that either uses HTTP GET or HTTP POST request can be used to send SMS.

You need to enter `smsprovider.HttpSMSProvider.HttpSMSProvider` as SMSProvider.

9.2.2 DeviceSMSProvider

LinOTP also comes with an SMSProvider module to enable the LinOTP server to send SMS itself. This can be achieved by connecting an appropriate phone to the LinOTP machine. The SMS are then send via this phone directly. You need to enter `smsprovider.DeviceSMSProvider.DeviceSMSProvider` as SMSProvider.

9.2.3 SmtpSMSProvider

LinOTP can also send SMS via SMTP-Gateways. This means an email is sent to a fixed email address with a certain subject and body. This is interpreted and the gateway sends an SMS to the phone number that is contained either in the subject or the body.

You need to enter `smsprovider.SmtpSMSProvider.SmtpSMSProvider` as SMSProvider.

9.3 SMSProviderConfig

Depending on the type of the SMSProvider, this configuration key stores different values.

9.3.1 HttpSMSProvider

For HttpSMSProvider this is:

```
{"URL":"http://smsgateway.com/sms_send_api.cgi",
 "PARAMETER": {
     "from":"0170111111",
     "password":"yoursecret",
     "sender":"name",
     "account":"company_ltd"
  },
 "SMS_TEXT_KEY":"text",
 "SMS_PHONENUMBER_KEY":"destination",
 "HTTP_Method":"GET",
 "PROXY":"http://username:password@your-proxy:8080",
 "RETURN_SUCCESS":"ID"
}
```

The `URL` holds the link to the API of the SMS Gateway. The `PARAMETER` can hold any number of fixed parameters, that allow you to use this SMS Gateway to send SMS. Usually this will at least be something like a username or an account and a password.

`SMS_TEXT_KEY` denotes, which parameter in the URL will hold the SMS text itself.

`SMS_PHONENUMBER_KEY` denotes, which parameter in the URL will be used to pass the mobile phone number to which the SMS should be sent.

`HTTP_Method` denotes, if the API of the SMS Gateway should be used with HTTP GET or HTTP POST request.

PROXY is an optional parameter that needs to be set when the LinOTP server accesses the internet via a proxy. This can be either HTTP or HTTPS and you may specify a username for authenticating to the proxy server.

RETURN_SUCCESS matches the reply of the SMS Gateway to indicate successful sending of the SMS. The first characters of the reply are compared to this definition.

RETURN_FAIL matches the reply of the SMS Gateway to indicate the failed sending of the SMS. The first characters of the reply are compared to this definition. The *RETURN_FAIL* is only evaluated if the *RETURN_SUCCESS* is not defined.

This is an example configuration for the clickatell SMS Gateway:

```
     { "URL" : "http://api.clickatell.com/http/sendmsg",
       "PARAMETER" : {
    "user":"username",
    "password":"askme",
    "api_id":"12980436"
  },
  "SMS_TEXT_KEY":"text",
  "SMS_PHONENUMBER_KEY":"to",
  "HTTP_Method":"GET",
  "PROXY" : "http://user:pass@192.168.100.250:8080",
  "RETURN_SUCCESS" : "ID"
}
```

Note: You need to use double quotes and not single quotes! Using single quotes will result in an error and sending SMS will not be possible.

9.3.2 DeviceSMSProvider

The DeviceSMSProvider module uses the tool gnokii [1] to send SMS. The configuration parameter looks like this:

```
{"CONFIGFILE":"/etc/linotp2/gnokiirc"}
```

where the value of CONFIGFILE reflects the location of the gnokiirc configuration file. A valid configuration file will look like this:

```
[global]
model = AT
port = /dev/ttyACM1
connection = serial
```

Note:

1. The LinOTP service account needs to have write access on the device defined in the port variable. In this case on /dev/ttyACM1. Usually this can be achieved by adding the LinOTP service account linotp to the group dialout.
2. There were problems reported using gnokii version 0.6.28. It is recommended to use version 0.6.29.

9.3.3 SmtpSMSProvider

The configuration for the SmtpSMSProvider looks like this:

[1] http://www.gnokii.org/

```
{ "mailserver" : "smtp.yourdomain.com",
  "mailsender" : "linotp@yourdomain.com",
  "mailuser" : "account-to-login-to-smtp-server",
  "mailpassword" : "somesecret",
  "mailto": "mailadress@smstp-gateway",
  "subject" : "<phone>",
  "body" : "This is your OTP-value: <otp>"
}
```

The parameters mailuser and mailpassword are optional. The tag <phone> needs to be put where the smtp gateway expects it, either in the subject or in the body. The <phone> tag will be replaced with the mobile phone number of the user. The tag <otp> will be replaced by the generated OTP value. The tag can be places in the subject or in the body.

9.4 SMSProviderTimeout

This configuration key holds the time that an OTP value that was sent via SMS is valid. This number is interpreted in seconds. The default value is 300 e.g. 5 minutes. That means that after a user requested a SMS OTP he will be able to use this OTP value within the next 5 minutes. If the SMS arrives later or the user will not use this OTP value within 5 minutes, it will not be valid anymore.

10 | Security Module

Starting with LinOTP 2.5 it is supported to store the encryption key in Hardware Security Modules. Encryption and decryption can be performed in such a hardware module. LinOTP implements a concept of a security module abstraction layer i.e. even the old encryption key stored at `/etc/linotp2/encKey` now is handled via a security module. New modules can be added easily. To define a module you need to configure this in the linotp.ini file. If you do not add anything, the old encryption key functionality is used.:

```
linotpActiveSecurityModule = lunasa
linotpSecurity.lunasa.module = linotpee.lib.security.pkcs11.Pkcs11SecurityModule
```

You can define several modules. LinOTP at the moment comes with a PKCS11 security module. To define, which module should be used the key linotpActiveSecurityModule is used which takes the identifier of the module. To define a new module, you use the key linotpSecurity.lunasa.module, where “lunasa” is the identifier or the name of the module and the key takes the Python module.

The following keys are configuration that depend on the chosen module. In this case the PKCS11 module.:

```
#Config depending on module
linotpSecurity.lunasa.library = libCryptoki2_64.so
linotpSecurity.lunasa.configHandle = 21
linotpSecurity.lunasa.valueHandle = 22
linotpSecurity.lunasa.tokenHandle = 23
linotpSecurity.lunasa.defaultHandle = 22
linotpSecurity.lunasa.slotid = 1
```

10.1 PKCS11 module and SafeNet LunaSA

`library` defines the PKCS11 so library in your filesystem.

`slotid` is the slot of the PKCS11 module to use.

The `configHandle`**,** `valueHandle` **and** `tokenHandle` are the handles within the slot of the corresponding AES keys to do the encryption and decryption of the OTP PIN, of configuration values, OTP keys and passwords. The defaultHandle is used, If one of the other Handles is not defined.

10.2 Password handling

Usually the PKCS11 device needs a password to access the slot. This password can either be defined in the linotp.ini file or needs to be passed to the LinOTP server after it has started. To define it in `linotp.ini` do it like this:

```
linotpSecurity.lunasa.password = YourPassword
```

To pass the password later to the LinOTP server you can use the `linotpadm.py` command line client:

```
% linotpadm.py --admin=admin --url=https://localhost -C securitymodule --module=default
      python yubikey module not available.
      please get it from https://github.com/Yubico/python-yubico if you want to enroll yubikeys
      No module named yubico
      Please enter password for 'admin':
      Please enter password for security module 'default':
      { u'status': True,
      u'value': { u'setupSecurityModule': { u'activeSecurityModule': u'default',
      u'connected': True}}}
```

To check the status of the security module you can do this:

```
% linotpadm.py --admin=admin --url=https://localhost -C securitymodule
      python yubikey module not available.
      please get it from https://github.com/Yubico/python-yubico if you want to enroll yubikeys
      No module named yubico
      Please enter password for 'admin':
      { u'status': True,
      u'value': { u'setupSecurityModule': { u'activeSecurityModule': u'default',
      u'connected': True}}}
```

11 | LinOTP as OpenID Provider

Starting with version 2.4 LinOTP can work as an Identity Provider for OpenID. The users may authenticate with their identity in the form:

https://linotpserver/openid/id/<user@realm (user@realm)>

The users will be redirected to

https://linotpserver/openid/login

to authenticate to LinOTP. Their authentication is stored within a cookie, which lifetime can be configured with `/etc/linotp2/linotp.ini` using the parameter:

```
linotpOpenID.CookieExpire = 3600
```

11.1 Comfortable identity links

To make the OpenID identity more comfortable for your users, you may use Apache's mod_rewrite module. To use the identity like https://linotpserver/oid/<username> add something like this to the LinOTP Apache configuration:

```
RewriteEngine on
RewriteLog  /var/log/apache2/rewrite.log
RewriteRule ^/oid(.*)   /openid/id$1    [R]
```

If you want to make it even easier, you should add a second FQDN e.g. in the default configuration of your Apache. Let us assume, your LinOTP runs on https://linotpserver but the server also listens on http://identities.com.

Then you can add the following rewrite rule to the configuration of identities.com:80:

```
RewriteEngine on
RewriteLog  /var/log/apache2/rewrite.log
RewriteRule ^/(.*)   https://linotpserver/openid/id/$1    [R]
```

Now your users can use the identities like

identities.com/<username>

12 | Retrieving OTP values

Starting with LinOTP 2.4 it is possible to ask the LinOTP server for the OTP values of given tokens. This can be done by accessing the API

https://linotpserver/gettoken/getotp

You can either pass a serial number or the username. This function is used to set the “Tagespasswort” for a given user. As this can be seen as a security issue, several steps need to be taken to activate this function.

- Not all tokens support the getotp function. At the moment only the DPW token supports this. I.e. even if you activate this function an HMAC token or an mOTP token will never return its OTP value.
- You need to configure the controller `/gettoken/getotp` in the apache config file linotp2. i.e. you need to create a new users file gettoken-api so that you can authenticate to this controller. You could also activate client certificate authentication for this controller to gain maximum security.:

```
<Location /gettoken>
     AuthType Digest
     AuthName "LinOTP2 gettoken"
     AuthDigestProvider file
     AuthUserFile /etc/linotp2/gettoken-api
     Require valid-user
 </Location>
```

- You need to activate this controller in the /etc/linotp2/linotp.ini file like this:

```
linotpGetotp.active = True
```

Finally you need to define policies, which user is allowed to retrieve how many otp values. For defining such a gettoken policy see section *Gettoken Policies* (page 59).

13 | Self service portal

LinOTP provides a self service portal that can be used by the user to

- enroll motp, HOTP, TOTP, Google authenticator token,
- disable tokens,
- reset OTP PIN,
- reset mOTP PIN.
- ...any many other actions.

The self service portal can be accessed via

https://your-linotp-server/.

For more information on the self service portal see part *LinOTP User Guide* (page 181).

Note: As long as no selfserivce policy is defined, users are not able to do anything in the selfservice portal. To define a selfservice policy read *Selfservice policies* (page 47).

13.1 Self service portal Imprint

You may define an imprint or contact information page for your self service portal. This can be different for each realm. Therefor you can create a different file and thus different information for each realm. The files have to be located at:

```
/etc/linotp2/imprint/<realmname>.imprint
```

This file may contain HTML code, so that you can add styles and links.

14 | Tools

LinOTP comes with some tools to ease system maintenance.

14.1 linotp-sql-janitor

The linotp-sql-janitor is part of the linotp-ee package. It will delete old entries in the SQL Audit log table. It can take a high and a low watermark. If the number of entries exceed the high watermark, old entries are deleted, so that only the <low watermark> newest entries remain.

This tool by default is run via cron once a day.

You can specify the watermarks using the command line (–high, –low), or entries in linotp.ini (linotpAudit.sql.highwatermark, linotpAudit.sql.lowwatermark). The default is set to high=10000 and low=5000.

14.2 linotp-tokens-used

The linotp-tokens-used is part of linotp-ee package. This in fact is a munin [1] plugin to monitor the total amount of managed tokens, the number of unassigned tokens and the number of inactive tokens.

Of course the output can be used to be dumped to any other file.

The output can be used to set up a dynamic pay per use licensing model.

14.3 linotp-backup

The linotp-backup is part of linotp-ee package. This shell script backs up the database and the encryption key to an PGP encrypted backup file.

14.4 linotp-restore

The linotp-restore is part of linotp-ee package. This shell script restores a previously backed up database and encryption key.

14.5 linotp-convert-token.pl

The linotp-convert-token is part of linotp-ee package. This script converts a Safeword dat seed file to an Aladdin/SafeNet XML file.

[1] http://munin-monitoring.org

14.6 linotp-decrypt-otpkey

The linotp-decrypt-otpkey is part of linotp-ee package. This script can be used for debug purposes. It helps to decrypt an OTP seed from the token database.

14.7 LinotpLDAPProxy.pm

The LinotpLDAPProxy.pm is part of linotp-ee package. This is a perl module to be used with an OpenLDAP server to setup an LDAP proxy, to perform LDAP binds (authentication) with OTP values.

15 | Backup and Restore

In any case there are two things that need to be backed up:

1. Token database
2. Encryption key

Of course it might make sense to backup other configuration files, if you made any changes to them. These might be:

1. /etc/linotp2/linotp.ini
2. /etc/apache2/sites-available/linotp2
3. /etc/freeradius/clients.conf
4. /etc/freeradius/*

But usually the above files will have only minor changes, which are easy to remember. The linotp-ee package comes with a backup and restore script. To backup your data run:

```
linotp-backup
```

The script will add a time stamp into the backup files, so that you will be able to have more than one backup. It will also ask you for a passphrase to protect the encryption key, so that the encryption key can be stored outside of the LinOTP server. To restore your data run:

```
linotp-restore <SQL-dump> <encKey-File>
```

15.1 Disaster Recovery

In case of a disaster recovery, when you need to setup a complete new LinOTP server you need at least the backup of the encryption key and the database file.

1. Install the LinOTP server with the install CD
2. Setup the LinOTP server accordingly.
3. Copy the encryption key and the the database backup to the server.
4. Run the command:

   ```
   linotp-restore <sql-file> <encKeyFile>
   ```

5. Your backup data might be a bit older. i.e. the HMAC counter of the tokens might be out of sync with the data in the database. So ask your users to go to the self service portal and resynchronize their tokens.
6. If you made any other changes to the FreeRADIUS server like the shared secrets in the clients.conf or to the Apache server, you need to modify those files accordingly.

16 | PCI DSS

The Payment Card Industry Data Security Standard is a policy framework that defines necessary requirements when dealing with credit card information. It defines a rule set divided into twelve main subjects. LinOTP helps to fulfill the following requirements of PCI DSS v2.0.

Requirement	Requirement text	LinOTP actions	
8.3	Incorporate two-factor authentication for remote access to the network by employees, administrators and third parties.	LinOTP provides a two factor authentication based on one time passwords with hardware tokens.	
8.4	Render all passwords unreadable during transmission and storage on all system components using strong cryptography.	Sensible data like OTP secrets and passwords are stored in a hashed or encrypted way.	
10.1	Establish a process for linking all access to system components to each individual user.	Using LinOTP you can identify each user by two factors.	
10.2	Implement automated audit trails for all system components to reconstruct the following events:	LinOTP provides a sophisticated audit trail, that can tell which user authenticated under which condition and which administrator performed which task on a certain token.	
10.2.1	All individual accesses to cardholder data	If the access to cardholder data is protected by LinOTP authentication, this will be logged within the audit trail.	
10.2.2	All actions taken by any individual with root or administrative privileges	All actions by LinOTP administrators are logged in the audit trail.	
10.2.3	Access to all audit trails	The access to the audit trail via the LinOTP API is logged in the audit trail. Direct access to the database must be audited in another way by the database system.	
10.2.4	Invalid logical access attempts	All LinOTP authentication requests are logged in the audit trail.	
10.2.5	Use of identification and authentication mechanisms	All LinOTP authentication requests (successful and failed) are logged in the audit trail.	
10.2.6	Initialization of the audit logs	The LinOTP SQL Audit Trail gets only initialized when the sequence number is set to 1.	
10.3	Record at least the following audit trail entries for all system components for each event	LinOTP Audit Trail can store several different information	
10.3.1	User Identification	When a user is authenticating, the username or the administrator name is logged	
10.3.2	Type of event	The LinOTP action is logged	
10.3.3	Date and time	A timestamp is logged	
10.3.4	Success of failure indication	The success status of the event is logged	
10.3.5	Origination of event	The name of the LinOTP server, where the action was performed is logged	
10.3.6	Identity or name of affected data, system component or resource	Depending on the action additional information is logged	
10.4 10.4.1 10.4.2 10.4.3	Using time synchronization technology, synchronize all critical system clocks and times and ensure that the following is implemented for acquiring, distribution and storing time: Critical systems have the correct and consistent time Time data protected Time settings are received from industry-accepted time sources	The LinOTP server uses NTP. The timestamps for the audit trail are generated on the LinOTP server and not on the database server.	
10.5	Secure audit trails so the cannot be altered.	The LinOTP audit trail entries get digitally signed.	
10.5.1	Limit viewing of audit trails to those with a job-related need.	The access to the audit trail can be restricted by defining access policies.	
10.5.2	Protect audit trails from unauthorized modifications.	The LinOTP audit trail entries are digitally signed.	
10.5.3	Promptly back up audit trail files to a	Audit trail can be written to an SQL	

17 | Background information and concepts

17.1 Users with no token

You may have users with no token, but who need to authenticate to the application that you are securing with LinOTP. In this case you can configure these uses in the FreeRADIUS server. You need to edit the file `/etc/freeradius/users` and add the names of the users like this:

```
user1           Auth-Type := Accept
administrator   Auth-Type := Accept
```

In this case the users will be accepted no matter what password they provide. Starting with LinOTP version 2.1 you may also use the Simple Pass token in this scenario. In current versions of LinOTP you can also use the system setting “Pass on user no token”.

17.2 UserIdResolvers and Realms

LinOTP implements a sophisticated concept of realizing multi-client capability.

LinOTP reads users from external user stores. These can be LDAP, Active Directory or Novell eDirectories or SQL servers or flat files like the UNIX /etc/passwd. These connections are configured using the UserIdResolvers.

LinOTP is also able to address any combination of as many of these UserIdResolvers as you need to.

Therefor each UserIdResolver has a unique name. Using the unique name of a UserIdResolver and the loginname, a user can be biuniquely identified.

Imagine a user “Administrator” who might be located in every Active Directory LinOTP is accessing. By putting together the loginname (Administrator) and the name of the UserIdResolver, all administrators can be distinguished easily.

Another possible scenario for using several UserIdResolvers might be, that selecting all users from a directory would be to time consuming. Several UserIdResolvers could be configured to make only a portion of the users from a huge Active Directory available to LinOTP. Lets say, that all your users are located in one Active Directory but you only need to use a subset of your users. You can of course define intelligent search pattern to get the user subset. But you can also configure several LDAP UserIdResolvers to access different organizational units in your Active Directory which is shown in *UserIdResolvers for different OUs in one Active Directory* (page 88).

User1 would be seen by LinOTP because the user is located in the OU, which is defined in UserIdResolver myOU1. User2 would be seen because the user is located in the OU, which is defined in UserIdResolver myOU2 and the user is in the group “Engineers”. User3 would be seen because the user is located in the OU, which is defined in UserIdResolver myOU3 and the user is member of the

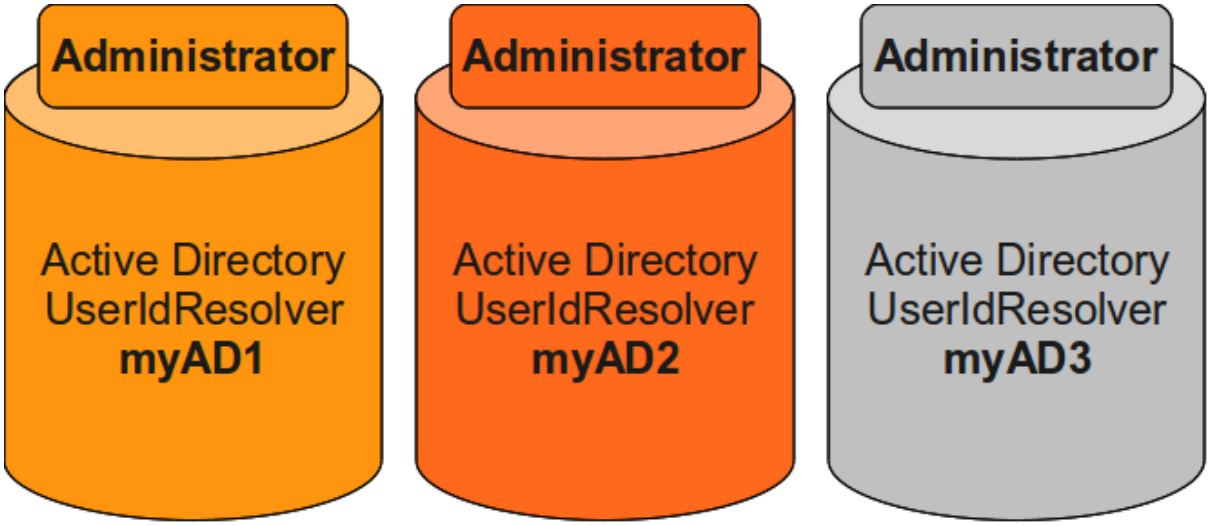


Figure 17.1: *UserIdResolvers for different Active Directories*

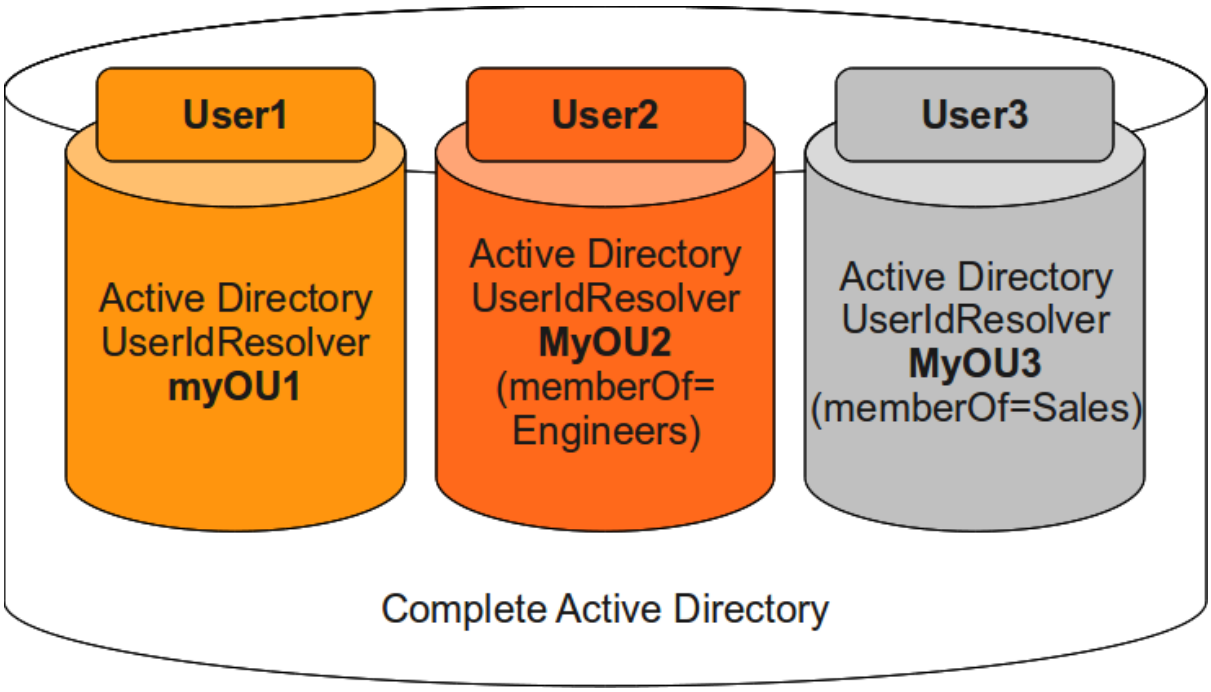


Figure 17.2: *UserIdResolvers for different OUs in one Active Directory*

group “Sales”. To be able to provide the previous scenario to different Active Directories, it is also possible to group several UserIdResolvers to realms. When authenticating the UserIdResolver and the realm can be passed along with the username to be able to identify the the user biuniquely.

17.2.1 Authenticating with Realms and UserIdResolvers

The LinOTP WebAPI, the PAM module and the FreeRADIUS module all can handle realms and UserIdResolvers. You may configure the realm and the UserIdResolver in the corresponding configuration files. This is described in part B Installation guide. Figure 30 gives you an idea of the flexibility of using realms and resolvers. The server FreeRADIUS 1 could be configured this way, that it always tries to authenticate the user1 only in the realm domain1. Users from other domains would not be able to authenticate to this RADIUS server. The server FreeRADIUS 2 does not have a special configuration. Users with the same username but from different domains can authenticate to this server. The user user1@domain1 (user1@domain1) will be found in the realm domain1 and the user user1@domain2 (user1@domain2) will be found in domain2. The server FreeRADIUS 3 will only authenticate users that can be found in the UserIdResolver “LDAP1”. All other users may not authenticate to this RADIUS Server.

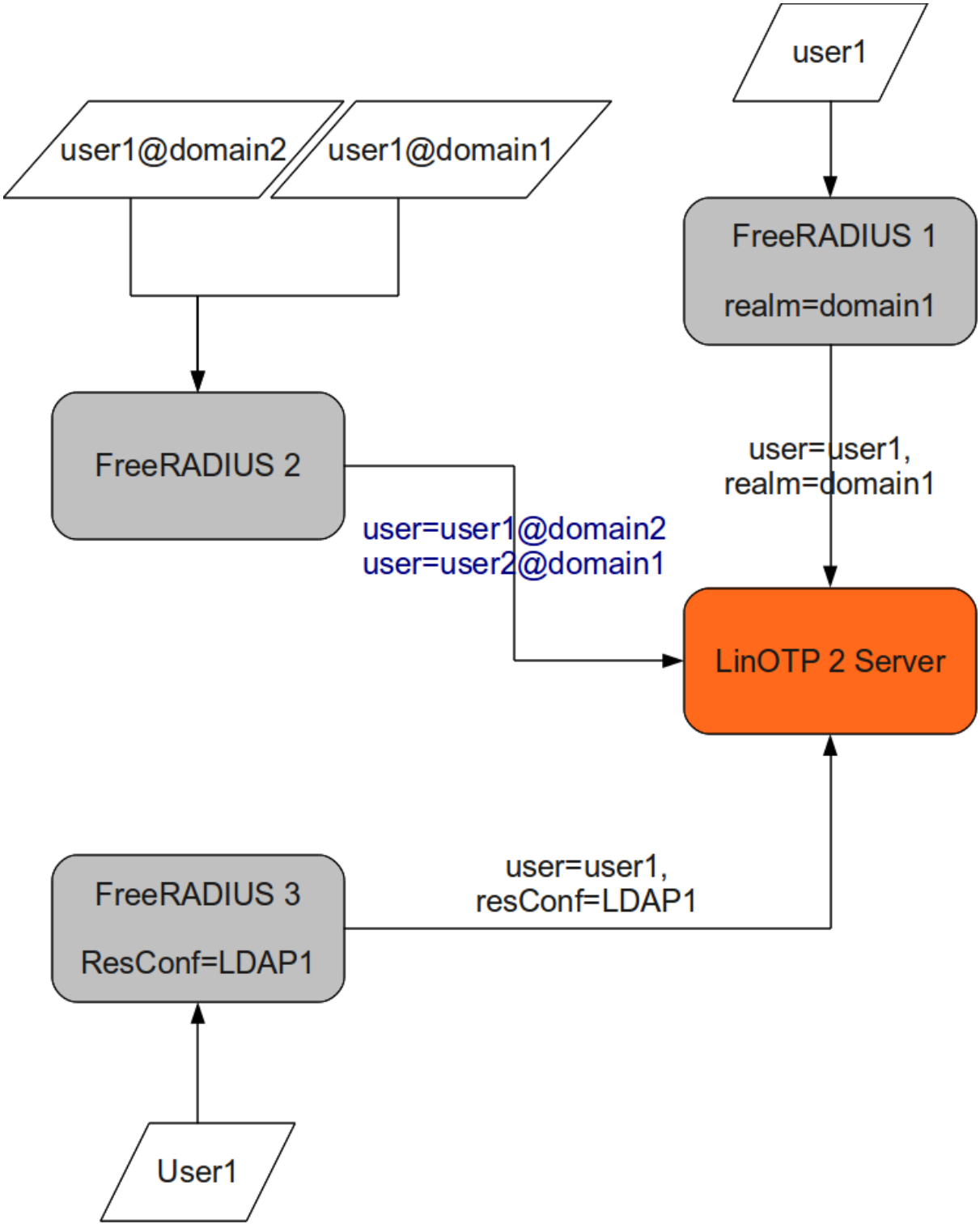


Figure 17.3: *Several possible Configurations using realms and UserIdResolvers*

18 | Usage scenarios

18.1 Protection levels

With LinOTP you can use different protection levels. This means you can assign different tokens to one person where this person has to use a hardware token to login to a secure environment (Application A) while the person can use his software token to login to a less secure environment (Application B).

This can be implemented by the use of different realms. You need to configure Application A that users from RealmA are allowed to login and Application B that users from RealmB are allowed to login.

Then define a UserIdResolver ResolverA pointing to this very user object. Also define a UserIdResolver ResolverB which will also point to the same user object. The ResolverB may be an exact copy of ResolverA just with the other name. Now put ResolverA into RealmA and ResolverB into RealmB.

Now you can use policy definitions to enroll hardware tokens in RealmA and software tokens in RealmB. Now the person has two tokens, but may only log on with the hardware token to the secure Application A and with the soft token to the less secure Application B.

19 | Troubleshooting

19.1 LDAP/Active Directory Connection

If you have got problems configuring your userstore and get your connection to the LDAP-Server or your Active Directory up and running, you may test the connection from the command line of the LinOTP server like this:

```
ldapsearch -h linotpserver -x -W -b 'ou=users, dc=az,dc=local' -D 'cn=admin,dc=az,dc=local' '(objectClass=user)
```

In case of Active Directory you will need to define objectClass=user, while in case of OpenLDAP or other LDAP implementations you may need to define other objectClasses.

19.2 Testing the Freeradius Server

The LinOTP Virtual Appliance also starts a RADIUS server that is accepting RADIUS requests from all RADIUS clients. To test the FreeRADIUS on your new installation you may enroll a token and issue the following command:

```
echo "User-Name=linotp,Password=042262" | sudo radclient -s <your-server-ip> auth <shared secret>
```

...where the Password is the OTP value of the token you assigned to the user linotp.

19.3 RADIUS Authentication fails

If RADIUS authentication fails or times out you should take a look at the FreeRADIUS log file usually located at `/var/log/freeradius/radius.log`.

A failed request would look like this:

```
Tue Sep 14 11:34:47 2010 : Error: rlm_linotp: (lotp_auth) LinOTPd returned ':-('
Tue Sep 14 11:34:47 2010 : Auth: rlm_linotp: Rejecting fall-through 'testuser'
```

To get more information about this you should now take a look at the LinOTP log file. (see section *OTP Authentication* (page 94)).

19.4 RADIUS server log file

You may monitor the FreeRADIUS server log file. The location may depend on your installation. If you are running the virtual appliance, it is `/var/log/freeradius/radiusd.log`.

A successful authentication will generate log entries like this:

```
rlm_linotp: (lotp_auth) LinOTPd returned ':-)'
rlm_linotp: user 'administrator' authenticated successfully
```

If the LinOTP server definitively denied access for the user, the log entry will look like this:

```
rlm_linotp: (lotp_auth) LinOTPd returned ':-('
Auth: rlm_linotp: Rejecting fall-through 'administrator'
```

In this case you should verify why the user is denied. As the RADIUS log does not give more details on "why" you should follow the troubleshooting in section A.19.5.

If the LinOTP server runs into some internal server error, a log entry like this will appear:

```
rlm_linotp: (lotp_auth) LinOTPd returned ':-/'
```

In this case you should take a deeper look into the LinOTP server log file or the Apache2 error log file on the LinOTP server. If the LinOTP server can not be contacted by the RADIUS server, such a log entry will show up:

```
Error: rlm_linotp: Error: ERROR: Error talking to linotpd server at \
https://localhost/validate/simplecheck?user=administrator&pass=test316988: couldn't connect to host
```

Verify that the LinOTP server can be reached (test it with PING, telnet or a browser) and check if the LinOTP server / Apache is running.

19.5 Access to Web API

If calling https://linotpserver/validate/simplecheck or https://linotpserver/manage/index returns an internal server error, please take a look at the apache error log file usually located at `/var/log/apache2/error.log`.

19.6 OTP Authentication

If OTP authentication fails for a user you might check the following:

- Verify that the tokens for this user are not locked.
- Verify that the failcounters of the tokens of the user have not reached the max failcount.
- Try to authenticate via the LinOTP Web API. This will give you a better idea of the error and rule out error source like the RADIUS server or the PAM module. For doing this use a webbrowser and go to the URL:

  `https://<yourlinotpserver>/validate/check?user=<login>&pass=<OTPPINOTPvalue>.`

- The return will be a JSON structure. Either download it and view it with your preferred text viewer or get the JSONViewer Plugin (https://addons.mozilla.org/de/firefox/addon/jsonview/) for your browser.
- Take a look at the log file on the LinOTP server. Probably it is the file: `/var/log/linotp/linotp.log`.
- Take a look at the RADIUS server log file.

19.7 LinOTP server log file

The LinOTP server log file is usually located at `/var/log/linotp/linotp.log` and is controlled by the settings in `/etc/linotp2/linotp.ini`. In the INI file you can configure the log level. A sensible setting

can be:

```
[logger_root]
level=INFO
[logger_linotp]
level=INFO
[logger_sqlalchemy]
level=WARN
[handler_file]
level=INFO
```

There are the following log levels available:

`DEBUG` for excessive debug out.

`INFO`

only a top level view of the things are logged in the INFO log level.

This can be successful authentication of users or also the information of changing the configuration of the LinOTP server by administrators. You may think of the INFO log level as some kind of auditing.

`WARNING`

the WARNING log level contains information where something went wrong,

but which is no malfunction of the server. For example the failing of authenticating a user triggers a WARNING log entry.

`ERROR`

events that indicate that something is wrong with your LinOTP server,

maybe some misconfiguration trigger an ERROR log entry.

`CRITICAL` not used at the moment.

In the following line the user “tester” failed to authenticate. The event is marked as a WARNI(NG).:

```
2010/12/01 - 14:57:44,074 WARNI [linotp.lib.token] [checkUserPass] user tester@ failed to authenticate.
```

The following line illustrates the successful authentication of the user “tester”. In this case an INFO event is written to the log file.:

```
2010/12/01 - 14:57:57,935 INFO  [linotp.lib.token] [checkUserPass] user tester@ successfully authenticated.
```

19.7.1 Special logging configuration

LinOTP uses the logging mechanism of the Pylons framework. You may read more about it here1. The logging mechanism is quite flexible. So you might log especially ERROR events to a different file or you might change the log level to DEBUG only for a certain LinOTP module, while all other modules are logging in the INFO log level.

You may log to the system event log or to TCP sockets or SMTP.

The following example adds a logger section onlylibs. The qualname defines the module, in this case linotp.lib.token. Thus only the token-module of LinOTP will write log entries in the DEBUG level. All other modules will write their log entries in the INFO level. Additionally these lib.token-Log entries are written to an additional file defined in the handler_file2 section. Thus you could use one log file that only contains INFO log entries for auditing that will be rotated only once a year and you can keep a DEBUG log file, that can be rotated every week or every day.:

```
[loggers]
keys = root, linotp, sqlalchemy , onlylibs
[logger_root]
```

```
level = INFO
handlers = file
[logger_linotp]
level = INFO
handlers = file
qualname = linotp
[logger_onlylibs]
level = DEBUG
handlers = file2
qualname = linotp.lib.token
[handlers]
keys = file, file2
[handler_file]
class = handlers.RotatingFileHandler
args = ('/var/log/linotp/linotp.log','a', 10000000, 4)
level = INFO
formatter = generic
[handler_file2]
class = handlers.RotatingFileHandler
args = ('/var/log/linotp/linotp-libs.log','a', 10000000, 4)
level = DEBUG
formatter = generic
[formatters]
keys = generic
[formatter_generic]
format = %(asctime)s,%(msecs)03d %(levelname)-5.5s [%(name)s] %(message)s
datefmt = %Y/%m/%d - %H:%M:%S
```

II LinOTP Installation Guide

1 | Supported Operating Systems

LinOTP will run on the following operating systems.

LinOTP Server (32 bit and 64 bit):

- Ubuntu 10.04+,
- Debian Squeeze via DEB packages,
- Debian Wheezy,
- RHEL 6.x,
- CentOS 6,
- OpenSuSE 11.2-12.3,
- Univention Corporate Server (UCS) 3.1.
- Other Linux distribution may run using the source package/pip installation method.

LinOTP Management Client:

- Ubuntu starting with 10.04 (32 and 64 bit),
- Debian Squeeze via DEB packages (32 and 64 bit),
- OpenSuSE 11.2 via source packages (32 and 64 bit),
- Windows XP 32 bit via integrated installer,
- Windows 7 32 bit and 64 bit via integrated installer,
- Windows Server 2008 and 2008 R2 via integrated installer.

1.1 Supported Database systems

LinOTP is known to run with the following Database systems:

- SQLite 3
- MySQL 5
- PostgreSQL 8, 9
- DB2 9
- Oracle 11g
- MS SQL Server 2005

2 | Checklist

Before installing LinOTP you should think about the following questions. This will make it easier for you to understand and plan your setup.

2.1 Decide which distribution you want to use

LinOTP server comes as a virtual appliance that is based on Debian Squeeze. It will install the operating system and all necessary components and do the basic configuration. LinOTP is also available as a ready made hardware appliance.

LinOTP server can also run on other Linux distributions but then the installation process requires some additional steps. Decide, if it is necessary for you to choose the installation on another distribution or if you will use the virtual appliance.

2.2 Decide which token database you want to use

Decide if you want to use the internal token database [1] on the LinOTP server (installed with the virtual appliance) or if you need to use an external database. You might have a dedicated database department that takes care of redundancy, availability and backups. The encryption key for the database stays on the LinOTP server, so that in case of using an external database no sensitive data will be exposed to another IT department. Have the database name and the username to access the external database at hand. For more information on the token database see *Setup token database* (page 110) and *Token database* (page 117).

2.3 SSL certificate

The communication to the LinOTP server is secured via SSL. During the installation a self signed certificate with SHA1 and 2048 bit RSA key will be generated.

You may also decide to replace this certificate with a certificate issued by a company CA.

2.4 LinOTP Admin

You need to pick a password for the LinOTP admin, who is allowed to manage the tokens. You may also create further administrative accounts or choose to authenticate using client certificates. This is accomplished by changing the file `/etc/apache/sites-enabled/linotp2`.

[1] The default internal database is a MySQL database

2.5 User databases

Determine where your users are located and get access to this database. If your users are located in an LDAP/Active Directory or in an SQL server, you will need to have a user with corresponding credentials to access this user database.

If your users are located in an LDAP or SQL server, you will also need to determine, which attribute or column holds the loginname.

2.6 User tokens

You need to decide, what tokens you are going to use. If you will use SafeNet eTokenNG OTP, you will need to install the SafeNet PKI Client and the native LinOTP management client - as these tokens need to be enrolled being plugged into the USB port.

2.7 Authentication

Where and how will the authentication be done? If you are using RADIUS, you need to configure your RADIUS clients accordingly. You might want to change the RADIUS shared secret by editing the file `/etc/freeradius/clients.conf` on the LinOTP server. When you are using the appliance you can use the web interface to configure the RADIUS clients and change the shared secrets.

2.8 Location in your network

The following figure shows the ports that LinOTP uses to communicate. This should help you if you want to place your LinOTP server within a separated subnet or demilitarized zone.

In figure *LinOTP workflow and network communication* (page 103) blue numbers represent steps for managing LinOTP and red numbers represent steps when authenticating with LinOTP. For managing and for authenticating the management clients and the authentication modules communicate with the LinOTP core on port 443 (blue 1 and red 2).

You usually will do the authentication via RADIUS. The LinOTP RADIUS plugin communicates with the LinOTP core on port 443, while the application (e.g. Firewall, SSL VPN) you are logging in to communicates to the FreeRADIUS server which by default is also located on the LinOTP server via Port 1812 and 1813 (red 1).

The LinOTP core communicates to the user database and the token database.

The user database will be an existing database like an SQL server or an LDAP server. So the LinOTP server needs to contact this LDAP server or Active Directory via port 389 or 636 (LDAP or LDAPs). If the users are stored within an SQL server, the LinOTP server needs to be able to reach those ports.

You may use the internal token database with the LinOTP server or you might want to use an external SQL server as your token database. In the later the LinOTP server needs to reach this external SQL database.

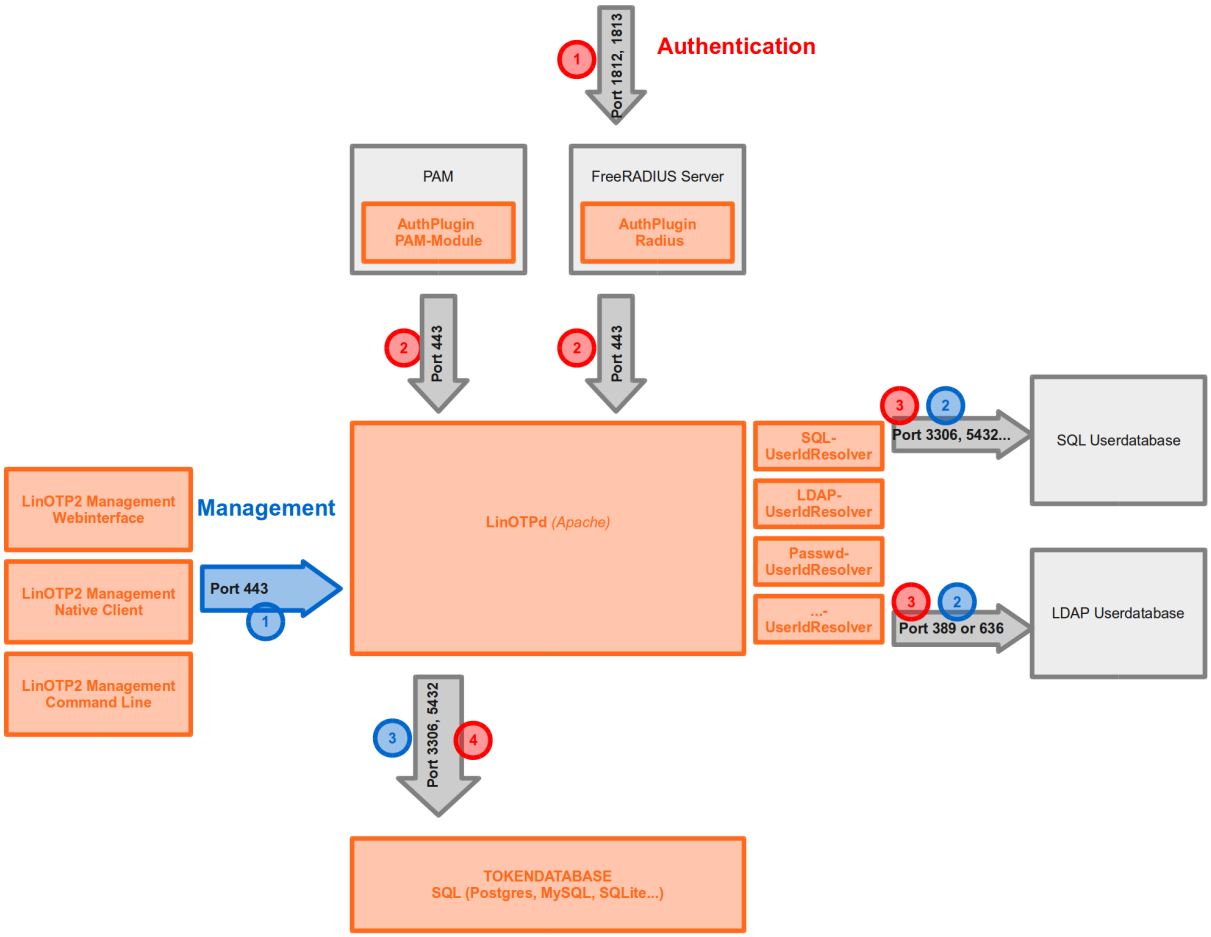


Figure 2.1: *LinOTP workflow and network communication*

3 | Server installation

You might want to install the different components on different machines. Only the LinOTP core and the UserIdResolver need to be installed on the same server machine. The authentication modules need to be installed on the authentication server, which might be a FreeRADIUS machine.

The LinOTP Management Clients can be installed on any machine, that you want to use to manage your LinOTP infrastructure.

Note: We recommend to install LinOTP Virtual Appliance as a single box using the ready made virtual install CD. Nevertheless, LinOTP Server can be also installed in other different ways on other Linux distributions. Either as tar.gz packages or as DEB packages. Please note, that the DEB packages will only work with Python 2.6 and Python 2.7. If your distribution features Python 2.5 you need to install the server the tar.gz way!

Note: After having installed the LinOTP server, you might want to decrease the loglevel by changing the file `/etc/linotp2/linotp.ini`. In the section [loggers] set all levels to WARN or INFO.

3.1 LinOTP Virtual Appliance installation

The Virtual Appliance is provided as a small install ISO image. This ISO image only contains iPXE code, the complete Appliance will be installed via the Internet from LSE repository servers. Thus when installing the Virtual Appliance you will always get the freshest software release.

During the installation process you will be asked for

- **serial number:** This is a serial number you get from your Reseller or Distributor. Based on this serial number you Virtual Appliance is granted access to the LinOTP software repositories.
- **keyboard layout:** You can choose the keyboard layout when logging in locally on the Virtual Appliance machine.

The Virtual Appliance can be installed either in a virtual machine or also on a dedicated computer.

Installing the Virtual Appliance takes the following steps:

1. Insert the ISO image and boot your computer form this CD.
2. On the welcome screen (*Welcome screen of the Virtual Appliance installer* (page 106)) you are asked, if you want to use DHCP or assign the IP configuration manually.

After the appliance is installed, it will still have the IP address you assigned manually to the Appliance (*Set the IP address of the Virtual Appliance manually* (page 106)).

Note: During the install process packages are loaded from linotp-appliance.lsexperts.de. Installing via a HTTP proxy is not supported.

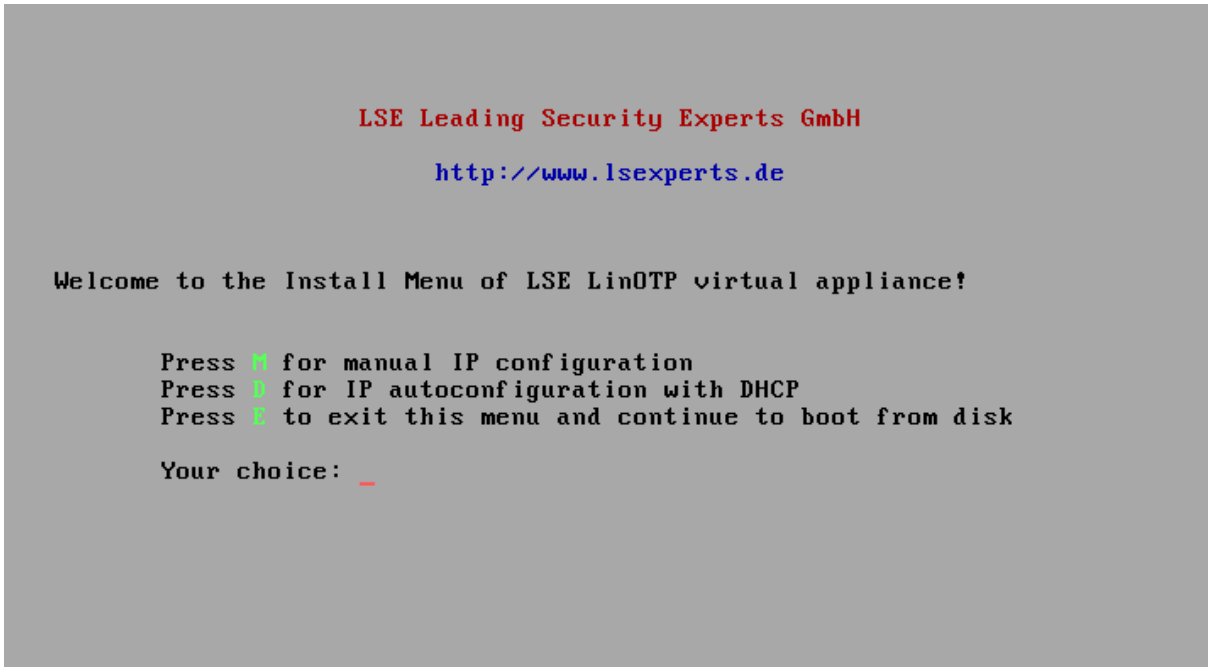


Figure 3.1: *Welcome screen of the Virtual Appliance installer*

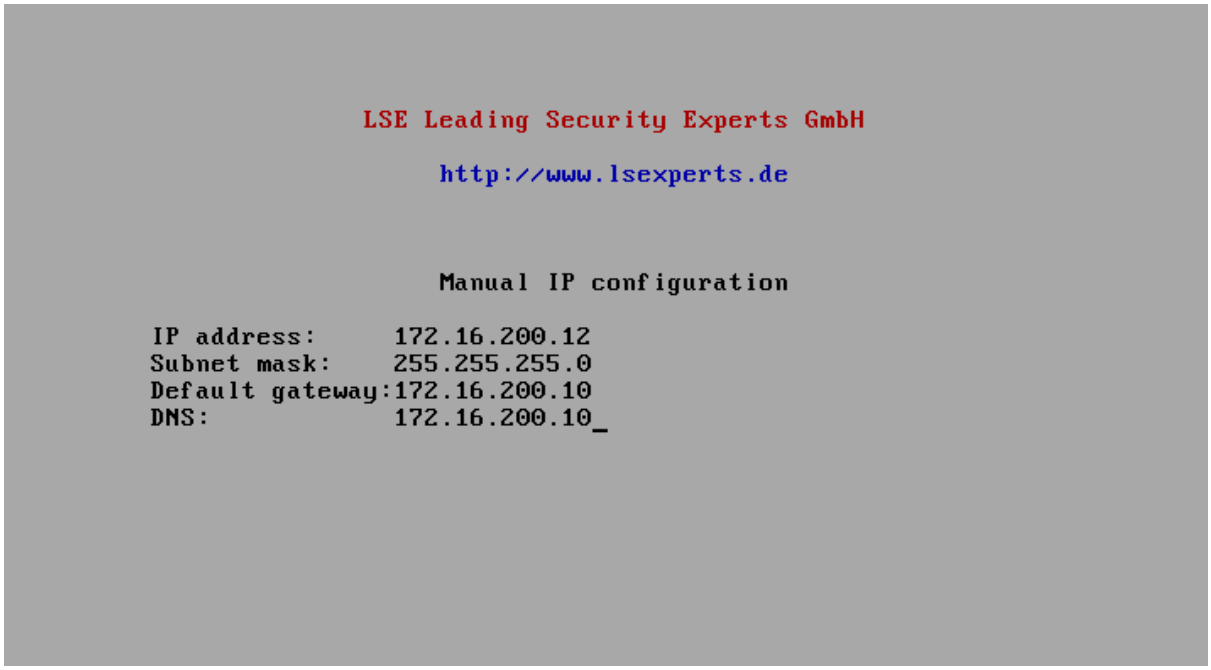


Figure 3.2: *Set the IP address of the Virtual Appliance manually*

3. After either choosing DHCP or setting the IP configuration you need to enter the serial number you received from your reseller or distributor. If the serial number is valid, the installation will start.

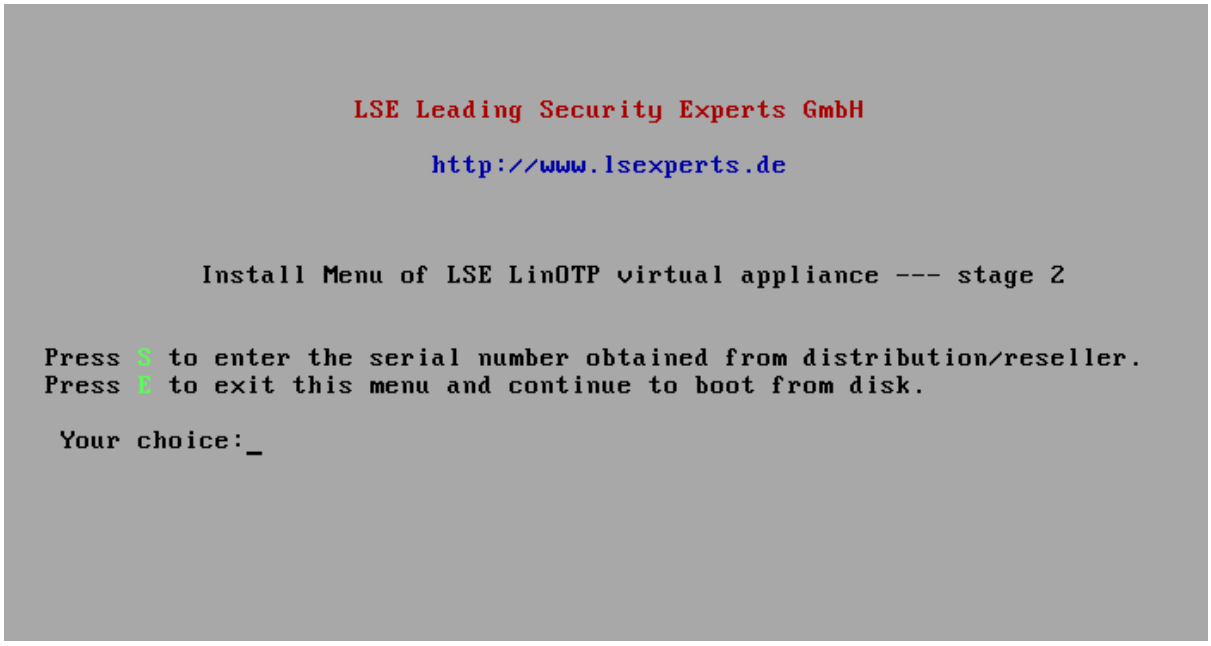


Figure 3.3: *Enter the serial number for the Virtual Appliance*

4. You are asked for the keyboard settings.
5. After the installation the system boots into your newly installed Virtual Appliance. Remove the CD/ISO to boot the system from the hard drive.
6. Now you can access your Virtual Appliance via a browser with HTTPS on port 8443 to run the setup wizard. Continue reading at *Getting started* (page 197).

3.2 Installing Debian packages

At the time of writing Debian packages are available that run with Ubuntu 10.04 LTS (32bit), Ubuntu 12.04 LTS (64bit) and Debian Squeeze (64bit). If you want to install LinOTP on another distribution, you should use the installation way described in *LinOTP Server Installation – the tar.gz, virtualenv and pip way* (page 109).

The LinOTP Debian packages depend on the following software you need to install:

```
apt-get install python-simplejson python-crypto python-docutils python-sqlalchemy \
                python-pylons python-tempita python-weberror python-webob python-mako \
                python-nose python-decorator python-formencode python-pastescript \
                python-pastedeploy python-paste python-beaker python-webhelpers \
                python-routes python-pygments python-m2crypto apache2 \
                libapache2-mod-wsgi python-repoze.who pwgen python-pyrad
```

If your users are located in an LDAP or Active Directory you also need to install the python LDAP package:

```
apt-get install python-ldap
```

If you will use MySQL or PostgreSQL as either tokenstore or userstore you need to install the corresponding DB module:

```
apt-get install python-mysqldb python-psycopg2
```

Install the LinOTP packages:

```
dpkg -i linotp_2.5.0_all.deb linotpdoc_2.5.0_all.deb \
        linotpuseridresolver_2.5.0_all.deb linotpuseridresolveree_2.5.0_all.deb \
        linotp-ee_2.5.0_all.deb smsprovider_2.5.0_all.deb
```

Some configuration screens will pop up, when installing the linotp package. This configuration will take care of creating the logging directory `/var/log/linotp` and

creating an encryption key `/etc/linotp2/encKey`.

Note: The encryption key should be backed up to a safe location.

You may rerun the configuration any time by issuing the command:

```
dpkg-reconfigure linotp
```

Note: During reconfiguration you are asked, if you want to create a new encryption key. If you do so, you will not be able to read your old data in the token database. But the encryption key is backed up to /etc/linotp2/encKey.old.

If you did not choose to create the token database during the configuration, you may now create the token database as described in section *Setup token database* (page 110).

3.2.1 Installing from APT repositories

The LinOTP components are also available via two apt repositories. You can use this method to install the LinOTP server and also the client components on Ubuntu 10.04 LTS (32bit) or 12.04 LTS (64bit).

Note: You can also use an APT repository to install LinOTP on Debian Squeeze. But as the Virtual Appliance is based on Debian Squeeze we recommend to install the Virtual Appliance, which will set up the repositories as needed.

You need to add these two lines to your `/etc/apt/sources.list` file:

```
deb http://linotp.org/apt/ubuntu lucid linotp2ce
deb https://<login>:<password>@customers.lsexperts.de/apt/linotp2/ubuntu lucid linotp2ee
```

The first repository contains all components of the community edition, the second repository contains the additional components of the enterprise edition.

For installing the the enterprise components, you need to specify the credentials you received with your download information. The packages are signed, using this key `https://customers.lsexperts.de/download/LinOTP2/LSE LinOTP2 Packaging linotp2@lsexperts.de (0xF86258E5%5C) pub.asc`.

Add it to your system:

```
apt-key add LSE\ LinOTP2\ Packaging\ linotp2@lsexperts.de\ \(0xF86258E5\)\ pub.asc
```

You need to enable apt to also talk to HTTPS repositories. Therefor you need to install an additional package:

```
apt-get install apt-transport-https
```

Now you need to tell apt to trust the root certificate that signed the certificate of the repository. You may download the CA certificate here:

```
http://www.trustcenter.de/certservices/cacerts/tc_class2_L1_CA_XI.crt
```

and put it to some directory. Now you need to create a file `/etc/apt/apt.conf.d/99linotp2` with the following content:

```
Acquire::https::CAinfo  "/your/directory/tc_class2_L1_CA_XI.crt";
```

to tell apt to trust this very CA certificate.

Now you may install the LinOTP server on your machine like this:

```
apt-get install linotpuseridresolver linotpuseridresolveree linotp linotp-ee
```

You may also install the LinOTP management clients from these repositories.

3.3 LinOTP Server Installation – the tar.gz, virtualenv and pip way

LinOTP can also run on many other distributions using the virtualenv for python. “virtualenv is a tool to create isolated python environments.” [1] This setup was run successfully on RedHat Enterprise Linux 6.2, 6.3 and CentOS 6. But due to the isolated environment it could run an many other distributions.

First you need to create a directory where the whole LinOTP installation will be located. We choose `/opt/LINOTP`. You might need to download the virtualenv tool here [2]

Create the virtualenv:

```
mkdir -p /opt/LINOTP
virtualenv --no-site-packages /opt/LINOTP
```

This will create an environment that does not use any python packages of your base linux installation.

Now enter this virtual environment:

```
cd /opt/LINOTP
source bin/activate
```

In fact this virtual environment is just a rewriting of several environment variables. You can leave this virtual environment any time by issuing the command `deactivate`. To be able to install all packages to this virtual environment you first need to install the following packages in your distribution using the required tool like apt-get, yum or yast...

- python-dev or python-devel
- swig
- gcc
- libssl-dev or openssl-devel
- openldap-devel or openldap-dev
- mysql-devel if you want to use mysql as token database.

Depending on your distribution the names may vary.

Now we are going to install all LinOTP stuff:

```
pip install linotp
```

...will install the LinOTP base packages with the base useridresolver from the python package index pypi.python.org. This public package index holds the same packages online as you might download from the LSE customer portal. If you prefer to install a manually downloaded linotp package you may as well issue a command `pip install linotp-2.5.0.tar.gz`.

Now you need to install further packages:

[1] http://www.virtualenv.org
[2] http://pypi.python.org/packages/source/v/virtualenv/virtualenv-1.7.1.2.tar.gz#md5=3be8a014c27340f48b56465f9109d9fa

```
pip install pyrad
pip install pil
pip install python-ldap
pip install repoze.who
pip install m2crypto
```

On some distributions (known with Ubuntu) m2crypto may fail to install. Then you can install python-m2crypto in your distribution and copy the corresponding files to `/opt/LINOTP/lib/python2.7/site-packages`.

Depending on where you want to store your token database you need to install database packages:

```
pip install pysqlite
pip install psycopg2
pip install MySQL-python
```

At this very moment you got a full working LinOTP Community Edition. To check which packages are installed in your virtual environment you can issue the command:

```
pip freeze
```

To make this to an Enterprise Edition you just need to install the following packages, that you need to download from the LSE customer portal:

```
pip install /path/to/packages/LinOTP-EE-2.x.tar.gz
pip install /path/to/packages/LinOtpUserIdResolverEE-2.x.tar.gz
pip install /path/to/packages/SMSProvider-2.x.tar.gz
```

Note: In this example we assume that you copy all config files to `/etc/linotp2/`. You could also use the directory `/opt/LINOTP/etc/linotp2` where some example files already exist.

Warning: The pip installation does not update itself automatically. LinOTP comes with a helper script `linotp-pip-update` which you can run to update your virtualenv. Of course you first need to enter the virtualenv before you can run the script.

3.3.1 Setup token database

You need to create the database, where the token information will be stored. This database can also be on a remote database server.

If you e.g. want to use mysql you could do it like this:

```
mysql -u root -p mysql
create database L2demo;
grant all privileges on L2demo.* to 'linotp'@'localhost' identified by 'mySecret';
flush privileges;
quit;
```

Note: The linotp package came with a config file `/etc/linotp2/linotp.ini.example`. You need to copy this file to your preferred location like `/etc/linotp2/linotp.ini`.

Change the following line accordingly:

```
sqlalchemy.url = mysql://linotp:mysecret@localhost/L2demo
```

You probably got no encryption key, yet. So generate a new encryption key:

```
dd if=/dev/urandom of=/etc/linotp2/encKey bs=1 count=96
```

Note: You can use the tool `linotp-create-enckey -f <linotp.ini file>` to create the encryption key.

Now you can create the database tables:

```
paster setup-app /etc/linotp2/linotp.ini
```

Check the ownership of the files

- `/etc/linotp2/encKey` (only readable for owner)
- `/etc/linotp2/token.db` (if used)
- `/var/log/linotp/` and `/var/log/linotp/linotp.log`.

These files should be owned by the service account `linotp`.

Note: You can fix ownership and access rights using the tool `linotp-fix-access-rights -f <linotp.ini file> -u <linotp user>`.

You can test this setup easily as described in *Testing LinOTP Server installation* (page 114).

3.3.2 LinOTP and the Apache webserver

More information about setting up LinOTP with Apache2 can be found in section *Apache2* (page 119).

The configuration depends on the distribution you are using. Debian based distributions split up the configuration in many different files for each site while RedHat based distributions use one big `httpd.conf` file.

With Apache we want to use authentication to the management web UI. For starters we choose to do digest authentication and create a file accordingly:

```
htdigest /etc/linotp2/admins "LinOTP2 admin area" admin
```

The following modules need to be installed and activated:

- mod_wsgi
- mod_ssl

Note: On systems using SELinux like RedHat 6 you need to deactivate SELinux to get LinOTP up and running. You may activate and reconfigure SELinux later on.

In `/etc/sysconfig/selinux` set `permissive` and reboot the machine.

On RedHat 6 based systems you need to create a directory, otherwise the WSGI module will fail with **Service Temporarily Unavailable**:

```
mkdir /var/run/wsgi
```

Now we can configure the apache server.

Things get easier, when we create an authentication file `/etc/[httpd|apache2]/linotp-auth.conf`, that can be included when need:

```
AuthType Digest
AuthName "LinOTP2 admin area"
AuthDigestProvider file
AuthUserFile /etc/linotp2/admins
Require valid-user
```

The configuration for your LinOTP site whether you configure it in httpd.conf directly or in an sites-enabled directory should contain the following. You might need to adapt directory names:

```
Listen 443
WSGIPythonHome /opt/LINOTP
WSGISocketPrefix /var/run/wsgi
<VirtualHost _default_:443>
   ServerAdmin webmaster@localhost
   DocumentRoot /var/www
   <Directory />
      Options FollowSymLinks
      AllowOverride None
   </Directory>
   <Directory /var/www/>
      Options Indexes FollowSymLinks MultiViews
      AllowOverride None
      Order allow,deny
      allow from all
   </Directory>

   Alias /doc/html         /usr/share/doc/linotpdoc/html
   WSGIScriptAlias /       /etc/linotp2/linotpapp.wsgi
   #
   # The daemon is running as user 'linotp'
   # This user should have access to the encKey database encryption file
   WSGIDaemonProcess linotp processes=1 threads=15 display-name=%{GROUP} user=linotp
   WSGIProcessGroup linotp
   WSGIPassAuthorization On

   <Location /admin>
      Include /etc/httpd/linotp-auth.conf
   </Location>
   <Location /audit>
      Include /etc/httpd/linotp-auth.conf
   </Location>
   <Location /gettoken>
      AuthType Digest
      AuthName "LinOTP2 gettoken"
      AuthDigestProvider file
      AuthUserFile /etc/linotp2/gettoken-api
      Require valid-user
   </Location>
   <Location /manage>
            Include /etc/httpd/linotp-auth.conf
   </Location>
   <Location /selfservice>
      # The authentication for selfservice is done from within the application
   </Location>
   <Location /system>
     Include /etc/httpd/linotp-auth.conf
   </Location>
   <Location /license>
     Include /etc/httpd/linotp-auth.conf
   </Location>
   <Location /validate>
      # No Authentication
   </Location>

   ErrorLog /var/log/httpd/error.log
   LogLevel warn

   # Do not use %q! This will reveal all parameters, including setting PINs and Keys!
   # Using SSL_CLIENT_S_DN_CN will show you, which administrator did what task
```

```
   LogFormat "%h %l %u %t %>s \"%m %U %H\"  %b \"%{Referer}i\" \"%{User-agent}i\" " LinOTP2
   CustomLog /var/log/httpd/ssl_access.log LinOTP2

   #   SSL Engine Switch:
   #   Enable/Disable SSL for this virtual host.
   SSLEngine on

   #   If both key and certificate are stored in the same file, only the
   #   SSLCertificateFile directive is needed.
   SSLCertificateFile    /etc/ssl/certs/linotpserver.pem
   SSLCertificateKeyFile /etc/ssl/private/linotpserver.key
   <FilesMatch "\.(cgi|shtml|phtml|php)$">
      SSLOptions +StdEnvVars
   </FilesMatch>
   <Directory /usr/lib/cgi-bin>
      SSLOptions +StdEnvVars
   </Directory>
   BrowserMatch ".*MSIE.*" \
     nokeepalive ssl-unclean-shutdown \
     downgrade-1.0 force-response-1.0

   ErrorDocument 500 "<h1>Internal Server Error</h1> Possible reasons can be missing modules or bad access righ
                     on LinOTP configuration files or log files. Please check the apache logfile \
                     <pre>/var/log/httpd/error_log</pre> for more details."

</VirtualHost>
```

Note: On RHEL and CentOS `mod_ssl` comes with a file `/etc/httpd/conf.d/ssl.conf` that already contains a VirtualHost definition. You also can adapts this accordingly.

Note: The example uses the SSL certificate linotpserver.pem. If you do not have a certificate authority that provides you the certificate, you might create a self signed test certificate as described in *Creating self signed SSL certificate* (page 114).

The WSGI daemon process will run as the user *linotp*. So you should have a system account or otherwise create it:

```
adduser -r linotp -d /opt/LINOTP
```

Copy the `linotpapp.wsgi` file from the local `etc` directory in your virtualenv to `/etc/linotp2` or adapt the `WSGIScriptAlias` in the apache configuration accordingly.

Check the permissions of the files

- `/etc/linotp2/linotp.ini` - `linotp` should have read access
- `/etc/Linotp2/encKey` - `linotp` should habe read access
- `/etc/linotp2/data/` - This is a template directory, `linotp` should have write access
- `/var/log/linotp/` - `linotp` should have write access

Note: You can add the user `linotp` and fix the access rights using the tool `linotp-fix-access-rights -f <linotp.ini file> -u linotp`.

You should now be able to restart the apache and login to the management web UI via `https://<yourserver>/manage`.

3.3.3 Further changes for RedHat and CentOS 6

You might want to adapt your firewall configuration in `/etc/sysconfig/iptables` to allow port 443 to be accessed.

Moreover you should verify, that your services `httpd` and `mysqld` (if you are using MySQL) are running after a reboot:

```
chkconfig mysqld on
chkconfig httpd on
```

3.3.4 Testing LinOTP Server installation

You are now ready to go. You might now want to start LinOTP temporarily from the command line:

```
paster serve /etc/linotp2/linotp.ini
```

LinOTP will now listen on the port you configured in the `linotp.ini` file.

If you configured Apache2 to serve LinOTP you just need to assure, that the apache server is started. Point your browser to `http://<yourserverIP>:5001/manage` and you will get the Management Interface.

> **Warning:** In the case of using paster serve, there is no authentication and encryption while talking to the LinOTP server. You should use this only for testing purposes! Please read the section *LinOTP and the Apache webserver* (page 111) and *Apache2* (page 119) on how to setup encryption and authentication.

3.3.5 Creating self signed SSL certificate

For testing purposes you can create a self signed certificate like this:

```
openssl req -new -x509 -key /etc/ssl/private/linotpserver.key \
        -out /etc/ssl/certs/linotpserver.pem -days 365
```

You should adapt the acces rights of the private key!

3.4 LinOTP on Univention Corporate Server UCS 3.1

LinOTP can also run on the Univention Corporate Server (UCS) 3.1. For this you can use the ready made LinOTP debian packages that can be downloaded from the LSE customers portal.

For the installation on the UCS we choose a local MySQL database as token database but you could also use an already running MySQL or PostgrSQL database somewhere in your network.

Note: You should install LinOTP on a UCS member server.

3.4.1 Install MySQL

Install the MySQL server with the following command:

```
univention-install mysql-server
```

3.4.2 Install dependencies

The following packages are needed by LinOTP but are not available via the maintained repositories.

python-docutils, python-pylons, python-tempita, python-weberror, python-webob, python-mako, python-nose, python-decorator, python-formencode, python-pastescript, python-pastedeploy, python-paste, python-beaker, python-webhelpers, python-routes, python-pygments, libapache2-mod-wsgi, python-repoze.who, python-pyrad, python-netaddr, python-configobj.

To install these packages we need to set:

```
ucr set repository/online/unmaintained="yes"
```

Now we can install all necessary packages:

```
univention-install python-simplejson python-crypto python-docutils \
python-sqlalchemy python-pylons python-tempita python-weberror \
python-webob python-mako python-nose python-decorator python-formencode \
python-pastescript python-pastedeploy python-paste python-beaker \
python-webhelpers python-routes python-pygments python-ldap python-m2crypto \
libapache2-mod-wsgi python-repoze.who pwgen python-pyrad \
python-mysqldb python-netaddr, python-configobj
```

If you like to, you can deactivate the installation of unmaintained packages again:

```
ucr set repository/online/unmaintained="no"
```

Note: If you deactivate unmaintained packages you also will not get any updates from the debian package repository!

3.4.3 Install LinOTP

Download the LinOTP packages from the customer portal [3].

And install the LinOTP packages:

```
dpkg -i python-qrcode_2.4.2-1_all.deb
dpkg -i linotp_2.5.2-1_all.deb linotp-ee_2.5.2-1_all.deb \
linotp-useridresolver_2.5.2-1_all.deb linotp-useridresolver-ee_2.5.2-1_all.deb \
linotp-smsprovider_2.5.2-1_all.deb
```

During the installation you will be asked several questions:

- Do you want to run LinOTP 2 via Apache2? **Yes**
- Password for LinOTP 'admin' account: <LinOTP admin password>
- Do you want to create a self signed certificate? **Yes**
- What SQL database do you want to use for the token database? **MySQL**
- Database hostname: **localhost**
- Name of the token database: **LinOTP2**
- Database user: **linotp2**
- Database users password: <choose a password>
- Do you want to create the LinOTP 2 database, now? **Yes**
- mysql root password: <empty!>

[3] https://customers.lsexperts.de/download/LinOTP2/2.5.x/squeeze-64bit/

Note: During installation no database root password was set, so you need to leave this empty!

3.4.4 Configure Apache2

As the Apache Port 443 is already used by UCS, you can configure a DNS Alias in the Univention Management Console like **linotp**. How this can be done, is described in the manual [4].

Now you need to configure the Named Virtual Host in the LinOTP apache config file `/etc/apache2/sites-enabled/linotp2`. Change the file like this:

```
ServerName linotp.<your domain>
ServerAlias linotp
```

Restart the Apache server:

```
service apache2 restart
```

3.4.5 Setup FreeRADIUS server

Most Firewalls and SSL VPNs use the RADIUS protocol for user authentication.

So now you need to install the FreeRADIUS server on your UCS:

```
univention-install freeradius
```

We need to open the port for FreeRADIUS in the firewall. So add the following to the file `/etc/security/packetfilter.d/50_local.sh`:

```
# LinOTP
/sbin/iptables -A INPUT -p "udp"  --dport 1812 -j ACCEPT
/sbin/ip6tables -A INPUT -p "udp"  --dport 1813 -j ACCEPT
```

and restart the firewall:

```
service univention-firewall restart
```

To configure FreeRADIUS for the use with LinOTP please see section *rlm_perl*.

3.4.6 Configure LinOTP

Now you need to create a UserIdResolver and your first realm.

You probably want to use your user from your LDAP. So you need to create a new LDAP resolver.

Note: The LDAP on the UCS is listening on port 7389, so you need to enter an LDAP-URI like ldap://<yourserver>:7389 or ldaps://<yourserver>:7636.

As BaseDN you should use `cn=users,dc=example,dc=com`.

As BindDN you can use `cn=Administrator,cn=users,dc=example,dc=com`.

Note: You need to replace `dc=example,dc=com` with your domain components.

You also need to set the Attributes like this to find your users in the UCS LDAP:

- LoginName Attribute: **cn**

[4] http://docs.univention.de/handbuch-3.1.html#ip-config:Konfiguration_der_DNS-Daten_in_der_Univention_Management_Console

- Searchfilter: **(cn=*)**
- Userfilter : **(&(cn=%s))**
- Attribute mapping: **{ "username" : "cn", ... }**
- UID Type: **dn**

Warning: You should create a service account and use this service account!

Please see *Configuring UserIdResolvers* (page 12) for further details on how to create your resolver and realm.

Now you are ready to enroll your first token and to authenticate.

3.5 Configuration background information

Note: This section contains deeper background information about server configuration. If you installed LinOTP via the virtual appliance or Debian packages, you probably do not need to read this.

You only need to read this chapter, if you installed LinOTP manually or want to adapt your LinOTP setup very specifically.

3.5.1 LinOTP Server configuration

Token database

The simple SQLite database may be well used for installations up to 100 tokens. This file based sql database can only be accessed locally. Being a simple file, it can be easily backupped but it features no additional access settings. Nevertheless all sensitive data are encrypted with the encKey file. But keep in mind, that this file is located on the same machine like the sqlite database file. You might have a dedicated database department that is running databases like Oracle or MS SQL. Please note, that LinOTP make use of the SQLAlchemy framework. You may find the supported databases here at http://www.sqlalchemy.org/docs/dialects/index.html.

You will then need to change the sqlalchemy.url in your `linotp.ini` file, which is usually located at `/etc/linotp2/linotp.ini`.

For Oracle change it to:

```
sqlalchemy.url = oracle://<user>:<password>@<servername>/<database>
```

For MS SQL change it to:

```
sqlalchemy.url = mssql://<user>:<password>@<servername>/<database>
```

Be assured that you set up the encKey before creating the database tables. The database tables are then created by calling:

```
paster setup-app /etc/linotp2/linotp.ini
```

A more detailed setup description for running LinOTP with Oracle or DB2 can be found at section *Database connection* (page 139).

Python search path

The path where your LinOTP components are installed must be contained in your python search path. This will be given, when you installed LinOTP by either of the described ways.

Anyway if the command `paster setup-app linotp.ini` returns the error message:

```
pkg_resources.DistributionNotFound: LinOTP
```

then your python search path needs to be fixed.

Then start a python shell:

```
# python
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', '/usr/local/lib/python2.6/dist-packages/docutils-0.5-py2.6.egg', '/usr/local/lib/python2.6/dist-packages/S
```

Check the output for your path. If it is not contained, you need to append your directory to the search path.

Press *Ctrl-D* to exit the python shell.

To append a new path to the Python search path, you need to place a file `linotp.pth` into an existing directory of the python search path. So create a file e.g. `/usr/lib/python2.6/site-packages/linotp.pth` with the following content:

```
/usr/local/lib/python2.6/dist-packages/
```

RADIUS token communication

Starting with LinOTP 2.4 you can also define RADIUS Tokens, that forward the authentication request to another RADIUS server. This can be another OTP backend or any other RADIUS server. In the file `/etc/linotp2/linotp.ini` you can define two parameters for this:

```
radius.dictfile= /etc/linotp2/dictionary
radius.nas_identifier = LinOTP
```

The *radius.dictfile* is the file that contains the RADIUS attributes definitions. LinOTP comes with a default one. But you can extend it according to special vendor attributes if necessary. The *radius.nas_identifier* is the NAS_Identifier that is also send within the RADIUS Auth Request. All other parameters are configured in the RADIUS token itself.

Please see the section *Enroll RADIUS Token* (page 36) for further information on this.

Paster

To get a quick result easily you may use the python-paste and just start the paster with the following command:

```
paster serve /etc/linotp2/linotp.ini
```

You should at least enable `ssl_pem` in your `linotp.ini` to have an encrypted communication between the authentication module or the management clients and LinOTP. Nevertheless, this will provide you with no authentication for the management, i.e. everyone having IP access to your LinOTP Server will be able to manage the tokens.

Note: *Paster* is also used to create the database tables (see *Token database* (page 117))

Start script

A Red Hat style start script is contained in the packages:

```
/ect/init.d/linotp2-paster
```

This will start the paster as user `linotp`.

Note: In the default configuration the paster would listen on all IP addresses, which means that the LinOTP server would be visible without encryption and authentication!

Note: When running with Apache2, you do not need this start script!

You can start the paster during boot time automatically by issuing the following commands:

```
chkconfig --add linotp2-paster
chkconfig linotp2-paster on
```

Apache2

To enable encryption and authentication for the LinOTP Server LinOTP can run in a web server. LinOTP will run as WSGI [5] and thus you can use any web server that supports WSGI. This section describes how to setup your Apache2 to serve LinOTP in an encrypted and authenticated manner.

1. Install apache2,
2. install libapache2-mod-wsgi [6],
3. enable mod-wsgi and mod-ssl with your apache:

   ```
   a2enmod wsgi
   a2enmod ssl
   ```

4. If you want to authenticate with username/password do a:

   ```
   a2enmod auth_digest
   ```

5. With LinOTP an example Apache2 configuration was installed to `/usr/share/doc/linotp/examples/apache2-linotp2`. You can copy this file to `/etc/apache2/sites-available/linotp2`.
6. Activate this configuration by:

   ```
   a2ensite linotp2
   ```

7. You may disable any other default site:

   ```
   a2dissite
   ```

In the following section we discuss the Apache2 configuration file, which you might want to adapt to your needs.

Apache2 configuration file

The sample configuration file:

[5] http://code.google.com/p/modwsgi
[6] This is the Debian package name. Might be different!

```
<VirtualHost _default_:443>
   ServerAdmin webmaster@localhost
   Alias /doc/html         /usr/share/doc/linotpdoc/html
   WSGIScriptAlias /       /usr/local/etc/linotp/linotpapp.wsgi
```

The *WSGIScriptAlias* binds the `linotpapp.wsgi` script to the web server root. You only should change the location of the `linotpapp.wsgi` script, but you should not change the web server root, since the corresponding web server paths are also included in the LinOTP Management Clients:

```
#
# The daemon is running as user 'linotp'
# This user should have access to the encKey database encryption file
WSGIDaemonProcess linotp processes=1 threads=15 display-name=%{GROUP} user=linotp
WSGIProcessGroup linotp
```

A distinct WSGI daemon process will be started to which the LinOTP wsgi will be delegated. This speeds up responses. Note, that the *user=<user>* needs to have write access to the log file specified in linotp.ini (default `/var/log/linotp/`) and read access to the `encKey` file specified in `linotp.ini`.

We now can setup the authentication for the different LinOTP interfaces. LinOTP provides the interfaces which reflect in the URL path:

/admin

This is used for all token management tasks like enrolling, assigning, deleting tokens. You should protect this interface either by password authentication provided by apache2 or by client certificate authentication. The Management Clients communicate via this interface.

/system

This interface is used to configure the LinOTP Server. You should protect this interface either by password authentication provided by apache2 or by client certificate authentication. The Management Clients (server configuration dialog) communicate via this interface. You can restrict access to this interface for a special user group, so that several users may manage the tokens but only a few users may configure server settings.

/validate

This interface is used to authenticate the users. The authentication modules `pam_linotp` and `rlm_linotp2` communicate via this interface. You should encrypt the communication with this interface (running apache ssl) and you may also setup authentication to this interface, but it is not that crucial as the /admin and the /system interfaces.

You might want to restrict the IP addresses that are allowed to contact this interface. This can be achieved by a configuration like:

```
<Location /validate>
   Order deny,allow
   Deny from all
   Allow from 192.168.1.17
</Location>
```

/license

This interface is used to load licenses to the LinOTP server and to retrieve license information. It should be protected in the same authentication manner like /admin or /system.

/selfservice

This interface is used for the enduser to manage his own OTP tokens. This interface provides its own authentication, so there is no special need to protect it within the apache configuration. The user is authenticated against the user database that is defined in the corresponding useridresolver. As the user authenticates with a username to this interface, /selfservice should only be accessible from your local network.

/manage

This interface is used for administrative tasks. All tasks that can be done with the graphical management client – except enrolling eToken NG-OTP – can also be done with this web interface. The authentication to this web interface needs to be setup in the Apache configuration. You may also setup OTP authentication for the management interface. For how to do this see *OTP Authentication with Apache2* (page 159):

```
<Location /admin>
#---------------------------------------
# We do this, when we do digest auth
#---------------------------------------
# AuthType Digest
# AuthName "LinOTP2 admin area"
# AuthDigestProvider file
# AuthUserFile /usr/local/pylons/etc/admins
# Require valid-user
#---------------------------------------
# Here we do client certificate auth
#---------------------------------------
    SSLVerifyClient require
    SSLVerifyDepth  2
  # Who signed the client certificates
    SSLCACertificateFile /etc/ssl/certs/ca.crt
  # what client certs are allowed to log in?
    SSLRequire ( %{SSL_CLIENT_S_DN_OU} eq "az" and %{SSL_CLIENT_S_DN_CN} in {"linotpadm", "Manfred Mann"}
</Location>

<Location /license>
   AuthType Digest
   AuthName "LinOTP2 admin area"
   AuthDigestProvider file
   AuthUserFile /usr/local/pylons/etc/admins
   Require valid-user
</Location>

<Location /selfservice>
    AuthType Digest
    AuthName "LinOTP2 self service"
    AuthDigestProvider file
    AuthUserFile /usr/local/pylons/etc/selfservice
    Require valid-user
</Location>

<Location /system>
#---------------------------------------
# We do this, when we do digest auth
#---------------------------------------
#       AuthType Digest
#       AuthName "LinOTP2 admin area"
#       AuthDigestProvider file
#       AuthUserFile /usr/local/pylons/etc/admins
#       Require valid-user
#---------------------------------------
# Here we do client certificate auth
#---------------------------------------
    SSLVerifyClient require
    SSLVerifyDepth  2
    # Who signed the client certificates
    SSLCACertificateFile /etc/ssl/certs/ca.crt
    # what client certs are allowed to log in?
    SSLRequire ( %{SSL_CLIENT_S_DN_OU} eq "az" and %{SSL_CLIENT_S_DN_CN} in {"linotpadm", "Manfred Mann"}
</Location>
<Location /validate>
    # No Authentication
```

```
    </Location>
```

The above Location directives demonstrate how to do authentication either with digest authentication or with client certificate authentication. To create a AuthUserFile for digest authentication you should use the command htdigest.

If you are using client certificate authentication you can restrict access for certain client certificates using the SSLRequire directive. For a guideline on how to setup SSL and to issue certificates see section *Issuing Certificate Authority* (page 123):

```
    ErrorLog /var/log/apache2/error.log
    LogLevel warn
    # Do not use %q! This will reveal all parameters, including setting PINs and Keys!
    # Using SSL_CLIENT_S_DN_CN will show you, which administrator did what task
    LogFormat "%h %l %u %t %>s \"%m %U %H\"  %b \"%{Referer}i\" \"%{User-agent}i\" \
        \"%{SSL_CLIENT_S_DN_CN}x\"""" LinOTP2
    CustomLog /var/log/apache2/ssl_access.log LinOTP2
    #   SSL Engine Switch:
    #   Enable/Disable SSL for this virtual host.
    SSLEngine on
    #   If both key and certificate are stored in the same file, only the
    #   SSLCertificateFile directive is needed.
    SSLCertificateFile    /etc/ssl/certs/lse-lap001.pem
    SSLCertificateKeyFile /etc/ssl/private/lse-lap001.key

    <FilesMatch "\.(cgi|shtml|phtml|php)$">
            SSLOptions +StdEnvVars
    </FilesMatch>
    <Directory /usr/lib/cgi-bin>
            SSLOptions +StdEnvVars
    </Directory>
    BrowserMatch ".*MSIE.*" \
            nokeepalive ssl-unclean-shutdown \
            downgrade-1.0 force-response-1.0
</VirtualHost>
```

Different authentication schemes for Management Interface

There are several different methods to authenticate to the management interface. All three methods are supported by all clients, the WebUI, the command line client and the GTK client.

Username/Password

This can be either basic authentication or digest authentication.

Note: If you want to authenticate against LDAP (see section *Authenticating users against LDAP or Active Directory* (page 123)), you need to do basic authentication.

In the GTK-Client you need to set “Basic” or “Digest” in the Client Configuration.

You can activate this by:

```
a2ensite linotp2
```

You need to deactivate other schemes by using `a2dissite`.

RADIUS Authentication

You can use the RADIUS authentication to authenticate with OTP tokens to the LinOTP Management.

Activate this by:

```
a2ensite linotp2-radius
```

You need to deactivate other schemes by using `a2dissite`.

In the GTK client you need to set “Basic Authentication” when using this scheme.

Client Certificates

You can also authenticate to the LinOTP management by using client certificates. Please note, that the LinOTP server then will use the Common Name of the Distinguished Name of the client certificate.

Using the WebUI the client certificate can also be located on a smartcard. Using the GTK-Client you need to configure the certificate file and the key file.

Note: Using the GTK-Client it is not possible to have the key file protected by a passphrase!

Activate this by:

```
a2ensite linotp2-certs
```

You need to deactivate other schemes by using `a2dissite`.

Authenticating users against LDAP or Active Directory

You may also want to authenticate to your LinOTP server with already existing accounts in your LDAP or Active Directory. To accomplish that, you need to configure the corresponding Location directive and do the following steps:

Activate `authnz_ldap` for Apache2:

```
a2enmod authnz_ldap
```

Configure the Location directive. This could be the /admin, /system and /license controller. To do this edit the file `/etc/apache2/sites-enabled/linotp2`:

```
 <Location /admin>
        AuthBasicProvider ldap
        AuthType Basic
        AuthName "LinOTP 2 admin area"
        AuthLDAPURL "ldap://172.16.200.61:389/ou=linotp,dc=linotp-test,dc=local?sAMAccountName?sub?(objectClass
        AuthLDAPBindDN "LDAPserviceUser@linotp-test"
        AuthLDAPBindPassword "test123!"
        Require ldap-group cn=linotp_admin_users, ou=linotp, dc=linotp-test, dc=local
</Location>
```

In this case all members of the group *linotp_admin_users* are allowed to manage LinOTP and its tokens. For other possible filters and restrictions see the *authnz_ldap* documentation at http://httpd.apache.org/docs/2.1/mod/mod_authnz_ldap.html .

To activate the changes restart Apache2:

```
/etc/init.d/apache2 restart
```

Issuing Certificate Authority

If you want to setup Apache2 to run with SSL you need to have a CA (certificate authority) certificate, a web server certificate and the private key to the web server certificate. If you do not have a CA and a web server certificate available, there is also a sample CA contained in the LinOTPdoc package.

You can find the CA depending on your installation at /usr/share/doc/linotp/examples/CA or /usr/local/share/doc/linotp/examples/CA.

The sample just consists of two files:

- Makefile
- openssl.cnf

Either change to this directory or copy the files to a location you like.

Prior to using the CA, you need to have your distribution packages make and openssl installed.

Typing:

```
make
```

gives you an overview, what you can do with this sample CA. To get started with LinOTP and Apache2 + SSL you need to do the following steps:

1. Create your new CA:

   ```
   make initca
   ```

   This will create a subdirectory keys and it will create the CA certificate. The file `keys/ca.crt` is the CA certificate for the apache directive SSLCACertificateFile.

2. Create a request for your web server:

   ```
   make req ext=server name=<servername>
   ```

3. Sign the request for the web server / issue the certificate for the web server:

   ```
   make sign ext=server name=<servername>
   ```

   This will create the file `keys/<servername>.pem` for the apache directive `SSLCertificateFile` and the file `keys/<servername>.key` for the apache directive `SSLCertificateKeyFile`.

4. Create a request for a admin user if you want to use client certificate authentication:

   ```
   make req ext=user name=<adminname>
   ```

5. Sign the request for the admin user / issue the certificate for the admin user:

   ```
   make sign ext=user name=<adminname>
   ```

6. The files `keys/<adminname>.pem` and `keys/<adminname>.key` need to be transferred to the client and configured in the LinOTP Management Client GUI.

   Note: The private key files created in this process are passphrase protected. If you need to remove the passphrase use:

   openssl rsa -in adminname.key -out adminname-wop.key

   ---

   which will create a key file adminname-wop.key without a passphrase.

Note: linotpadm.py supports key files with passphrases at the command line. glinotpadm.py does not handle passphrases nicely at the moment. The request for the passphrase will be echoed to the terminal, where you started glinotpadm.py. The passphrase will be requested for every connect to the LinOTP Server, which naturally are a lot. Digest authentication and client certificate authentication can not be used at the same time.

---

4 | Installing Management Clients

The Management Clients are also implemented in Python and can be used on Linux and on Windows. There are DEB packages available which are known to run on Ubuntu 10.04 LTS (32bit), 12.04 LTS (64bit) and Debian Squeeze (64bit). If the package fails to run on your system, you can also install the management client from source or from the python package index (see *Install python packages* (page 125)).

The Management Client also runs on Windows XP, Windows 7 and Windows Server 2008.

Note: There are two admin clients available – a command line client linotpadm.py and a GTK graphical user interface glinotpadm.py. The GTK client will only work with Python 2.6. The command line client can very well work with Python 2.5. The command line client only provides a reduced feature set.

Note: If you want to enroll the USB based eToken NG-OTP you also need to install the SafeNet PKI Client. If you do not want to enroll eToken NG OTP and not enroll Yubikeys you can do all management task with the Web UI and do not need to install the management client.

After having installed the management client, please refer to *LinOTP Management Guide* (page 5) for configuring your LinOTP server and using the management client.

4.1 Install the Debian packages

The DEB packages for the Management Client require Python 2.6 to be installed on your system. On a distribution like Ubuntu 10.04 LTS, 12.04 LTS or Debian you may install the Management Client by issuing the command:

```
dpkg -i linotpadminclientce_2.5.0_all.deb linotpadminclientee_2.5.0_all.deb
```

If your Python search path is setup correctly (see *Python search path* (page 117)) you may now start the management client:

```
glinotpadm.py
```

4.2 Install python packages

On other distributions you can install the management client either system wide or to your virtualenv:

```
pip install linotpadminclientce
pip install /path/to/package/LinOTPAdminClientEE-2.4.4.tar.gz
```

4.3 Management Client Installation on Windows

On a MicrosoftR Windows you simply may install the bundled full installer `LinOTP2MgmntClient-2.5.2-Full-Installer.exe`.

The Management Client is known to run on Windows XP, Windows 7 (32/64bit), Windows 2008 and Windows 2008R2.

The following software is contained in the bundle installer:

1. http://www.python.org/ftp/python/2.7.4/python-2.7.4.msi (You will not necessarily need Tcl/Tk, Documentation and Test suite)
2. http://ftp.gnome.org/pub/GNOME/binaries/win32/gtk+/2.18/gtk+-bundle_2.18.7-20100213_win32.zip
3. http://ftp.gnome.org/pub/GNOME/binaries/win32/pycairo/1.8/pycairo-1.8.6.win32-py2.6.exe
4. http://ftp.gnome.org/pub/GNOME/binaries/win32/pygobject/2.20/pygobject-2.20.0.win32-py2.6.exe
5. http://ftp.gnome.org/pub/GNOME/binaries/win32/pygtk/2.16/pygtk-2.16.0.win32-py2.6.exe
6. http://chandlerproject.org/pub/Projects/MeTooCrypto/M2Crypto-0.19.1.win32-py2.6.exe

Just run the installer and follow the instructions.

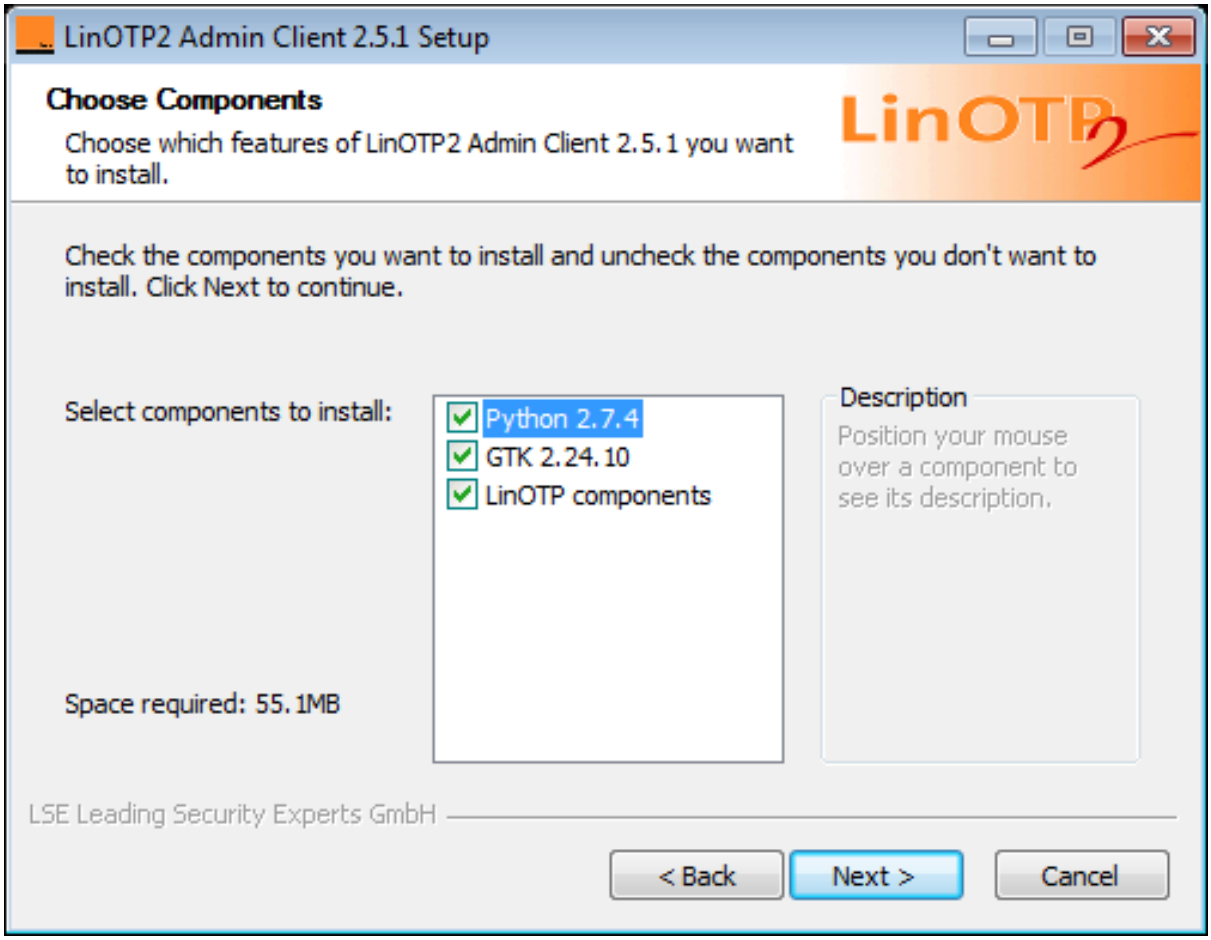


Figure 4.1: *Choosing the components to install*

Note: If you want to enroll the USB based eToken NG-OTP you also need to install the SafeNet PKI Client.

Note: If you want to change the language of the LinOTP Management Client after the installation, you need to set the environment variable *LANG* to either *de* or *en*.

4.3.1 Installing Yubikey components

You need to install several additional components to enable the USB support and enroll Yubikeys on this workstation.

Note: You do not need to install any software to use the Yubikey.

Install setuptools

Download setuptools [1] and execute the installer.

Note: As the full installer comes with python 2.7 32bit, you also need to install setuptools 32bit even if you are on a 64bit system.

Python Yubico module

Download the modified python Yubico module [2]. The original module will not work at the moment.

Unpack it and open a command prompt as administrator.

In the unpacked directory run:

```
python setup.py install
```

Python USB

This are the python USB modules. Download [3], unpack it and open a command prompt as administrator.

In the unpacked directory run:

```
python setup.py install
```

Install the driver

Now you need to install a libusb driver for the Yubikey. There are several ways to do this, the easiest is to download the tool *zadig* [4], which will create the driver package for you.

You need 7zip to unzip the tool!

Download the tool and run zadig.

Insert a Yubikey and select this USB device (vendor ID 1050, product ID: 0010). You might need to check `List all devices` in the options menu.

The `Target Driver` needs to be set to `libusb0`. Hit install and wait.

[1] https://pypi.python.org/packages/2.7/s/setuptools/setuptools-0.6c11.win32-py2.7.exe
[2] https://github.com/lsexperts/python-yubico
[3] http://downloads.sourceforge.net/project/pyusb/PyUSB%201.0/1.0.0-alpha-3/pyusb-1.0.0a3.zip
[4] http://sourceforge.net/projects/libwdi/files/zadig/zadig_v1.1.1.140.7z/download

Note: When authenticating the Yubikey works as a HID device. This is why you do not need any additional driver installed. But for enrolling the Yubikey on your Windows machine the associated driver for the Yubikey is set to libusb0. Thus you will not be able to use the Yubikey for authentication on your enrollment workstation.

Finally you should reboot your machine.

5 | Installing Authentication Modules

An authentication module will talk a standard authentication protocol to the application at which the user shall authenticate and on the other side it will talk the HTTP protocol to the LinOTP Server. The authentication modules communicate to the LinOTP Server via interfaces like `/validate/check` and `/validate/simplecheck`. For more information on the interfaces see *Authentication interfaces* (page 231).

At the moment there are the following authentication modules available:

`pam_linotp`

: which can be plugged into the Unix PAM stack,

`rlm_linotp2`

: which plugs into FreeRADIUS server to enable all RADIUS-Application to authenticate against LinOTP.

Alternatively there is the integration into the Radiator RADIUS Server, a script for the `rlm_perl` module to run with FreeRADIUS [1] and a module to integrate LinOTP into simpleSAMLphp.

On the Internet you can also find a module to use LinOTP with Wordpress [2] and a module to do two factor authentication with Apache2 [3].

pam_linotp and rlm_linotp2 are implemented in C and need to be compiled for your platform. The FreeRADIUS module is also available as ready made DEB package compiled for Debian Squeeze 64bit.

5.1 Enabling PAM authentication with pam_linotp

The pam_linotp is contained in the package `LinOTP-2.x-CE.tar.gz`. Unpack the package and change to the directory `auth_modules`. Unpack `LinOTPAuthCE.tar.gz` and change to `LinOTPAuthCE/pam/src/`. Run the commands:

```
libtoolize
aclocal
automake -add-missing
autoconf
./configure
make
make install
```

This will install the PAM module to `/lib/security/pam_linotp.so`. You are now ready to use pam_linotp within your PAM stack.

[1] http://www.howtoforge.com/how-to-use-freeradius-with-linotp-2-to-do-two-factor-authentication-with-one-time-passwords
[2] http://wordpress.org/extend/plugins/linotp/
[3] https://github.com/lsexperts/mod_authn_linotp

Note: Depending on your distribution you will need to install a package like libpam-devel or libpam0g-dev to be able to compile the PAM module.

5.1.1 Configure pam_linotp

pam_linotp can be used in any way in the pam auth facility. If you want to be able to switch OTP authentication easily on and off for each service and your distribution supports PAM includes like `/etc/pam.d/auth-common`, you may adapt the auth-common easily and create a file `/etc/pam.d/auth-linotp`:

```
auth [success=1 default=ignore] pam_linotp.so debug url=https://localhost/validate/simplecheck
auth    requisite                       pam_deny.so
auth    required                        pam_permit.so
```

pam_linotp knows the following parameters:

`debug`

pam_linotp supports the debug flag to produce debug output.

`url`

The url parameter defines, where your LinOTP server is located. Leave /validate/simplecheck unchanged unless you exactly know, what you are doing.

`nosslhostnameverify`

The hostname of the LinOTP Server is not checked to match the common name in the certificate when doing SSL.

`nosslcertverify`

The validity of the certificate of the LinOTP Server is not checked.

`realm=<yourRealm>`

The user to authenticate is only searched in this realm. This parameter is optional. Here you may enter the name of one of the realms you configured.

`resConf=<special useridresolver>`

The user to authenticate is only searched in this UserIdResolver. This parameter is optional. Here you may enter the name of one of the UserIdResolvers you configured.

`loguser`

If set to “no”, no usernames will be logged in the log file. The default is “yes”.

`logpassword`

If set to “yes” also passwords will be logged in the log file. The default is “no”.

5.2 Enabling RADIUS authentication with rlm_linotp2

rlm_linotp2 is a authentication module for the FreeRADIUS server. The rlm_linotp2 can not be compiled alone. When introducing new modules to FreeRADIUS you need to compile the complete FreeRADIUS server. Thus you also need to get the rlm_linotp2 as source code.

1. Get FreeRADIUS 2.1.x – either as source from the project [4] or as source packages from your distribution.
2. For compiling FreeRADIUS and rlm_linotp2 you need at least gcc, make, libtool, libssl-dev (or openssl-devel) , libcurl-dev (or libcurl-devel) and depending on

[4] http://www.freeradius.org

your module selection further libraries.

3. Unpack the FreeRADIUS source.
4. Unpack LinOTP-2.x-EE.tar.gz.
5. You need to copy the source directory of rlm_linotp2 `auth_modules/LinOTPAuthEE/freeradius/src/rlm_linotp2`

   to the source directory of the FreeRADIUS modules. All modules reside in `freeradius-server-2.1.x/src/modules/`.
6. Within the FreeRADIUS source directory you need to add rlm_linotp2 to the file `src/modules/stable`.

   This file holds all modules that will be compiled with FreeRADIUS.
7. For compiling the FreeRADIUS please refer to the FreeRADIUS documentation. But you should be good with:

   ./configure && make clean && make
8. check for the shared library files being created `rlm_linotp2-2.1.8.so`, `rlm_linotp2.so`.
9. You may now install the FreeRADIUS Server:

   make install

5.2.1 Configure rlm_linotp2

The source directory of rlm_linotp2 contains an example configuration file linotp2.conf which needs to be copied to your FreeRADIUS config directory (usually /etc/freeradius):

```
linotp2 {
   validateurl = https://localhost/validate/simplecheck
   sslhostnameverify = yes
   sslcertverify = yes
   realm = <yourRealm>
   resConf = <special UserIdResolver>
}
```

The valid options are:

`validateurl`

The url parameter defines, where your LinOTP server is located. Leave /validate/simplecheck unchanged unless you exactly know, what you are doing.

`sslhostnameverify`

Whether the hostname of the LinOTP Server is checked to fit the common name in the certificate when doing SSL.

`sslcertverify`

Whether the validity of the certificate of the LinOTP Server is checked.

`realm=<yourRealm>`

The user to authenticate is only searched in this realm. This parameter is optional. Here you may enter the name of one of the realms you configured.

`resConf=<special useridresolver>`

The user to authenticate is only searched in this UserIdResolver. This parameter is optional. Here you may enter the name of one of the UserIdResolvers you configured.

`restrictusername`

If set to “yes” the username is checked if it only contains unharmful characters (letters, number and -_+.@). If set to “no” the characters in the username are not checked. This is possibly more dangerous but also allows to use unicode characters in the username. To test the FreeRADIUS Server setup you may issue a command like this:

```
echo "User-Name = koelbel@LOCAL, User-Password = 845788" | radclient -s -x 127.0.0.1 auth testing123
```

`loguser`

If set to “yes” the username is written to the log files. (default=yes).

`logpassword`

If set to “yes” the password is written to the log files. (default=no).

`allowemptypassword`

If set to “yes” the password can be emptry. (default=no).

5.2.2 Using rlm_linotp2 and SSL

If you are using `sslcertverify = yes` the certificate of the LinOTP server will be check. For this the FreeRADIUS needs to access the complete certificate chain of the root CA, possible intermediate CAs and issuing CA. You need to put all these certificates into the folder /etc/ssl/certs and need to create hash links [5] of those new certificates. The certificates need to be in PEM format and should have the file suffix .pem or .crt. You may use the command from the openssl package to create the hash links:

```
c_rehash /etc/ssl/certs
```

5.2.3 Fail-over configuration for RADIUS

The rlm_linotp2 module supports the fail-over configuration of the FreeRADIUS server. For more information on this see the FreeRADIUS documentation [6].

The fail-over configuration will work like this, that you may set up the FreeRADIUS server this way, that the rlm_linotp2 may talk to any number of LinOTP servers in a round robin like manner. So a simple redundant setup might look as in *The RADIUS failover setup.* (page 133).

To configure such a redundant setup you need to modify two configuration files of the FreeRADIUS server. In the file `/etc/freeradius/modules/linotp` you define instances of the rlm_linotp2 module. Create an instance for each server like this:

```
linotp2 LinOTPinstance1 {
       validateurl = https://linotp1/validate/simplecheck
       sslhostnameverify = yes
       sslcertverify = yes
}
linotp2 LinOTPinstance2 {
       validateurl = https://linotp2/validate/simplecheck
       sslhostnameverify = yes
       sslcertverify = yes
}
linotp2 LinOTPinstance3 {
       validateurl = https://linotp3/validate/simplecheck
       sslhostnameverify = yes
       sslcertverify = yes
}
```

Then you need to modify the file `/etc/freeradius/sites-enabled/linotp` and define, that the *AuthType LinOTP2* should use all these instances:

[5] A hash link is a link to a file, where the link name is the hash value of the file itself.
[6] http://wiki.freeradius.org/Fail-over

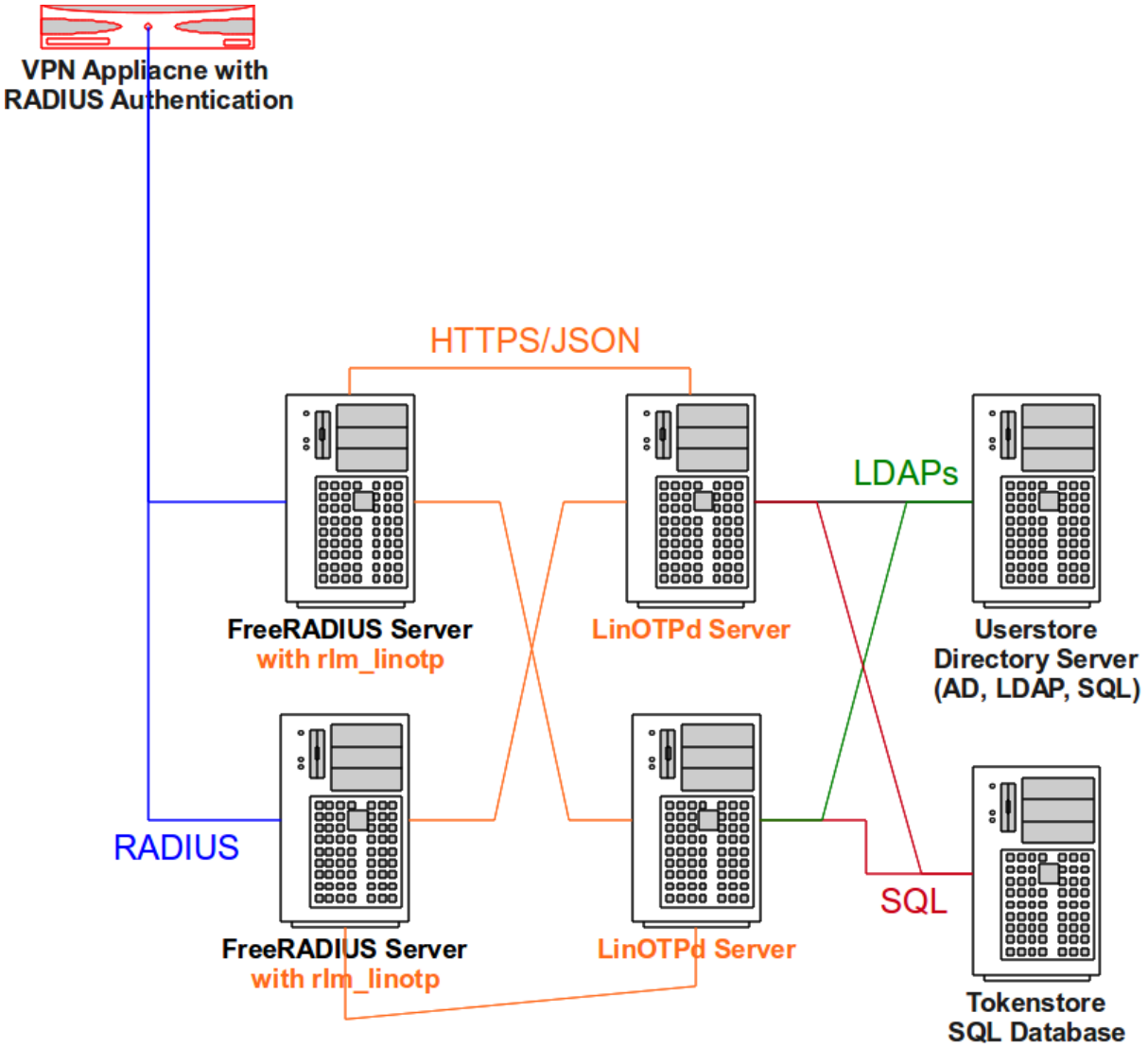


Figure 5.1: *The RADIUS failover setup.*

```
authenticate {
   Auth-Type LinOTP2 {
       redundant {
           LinOTPinstance1
           LinOTPinstance2
           LinOTPinstance3
       }
   }
[...]
}
```

5.3 Enabling RADIUS authentication with Radiator

Radiator [7] is a commercial highly configurable and flexible RADIUS server. It is written in Perl and thus can run on Linux, Unix, Mac OS X, Windows and VMS. Plugging LinOTP and Radiator together is fairly simple. Contact Open System Consultants Pty Ltd [8] to get your evaluation copy of Radiator.

To access LinOTP you will use the Authenticate Module *AuthUrl*. The Radiator configuration file could look like this:

```
<AuthBy URL>
 Identifier LinOTPURL
 Debug 1
 AuthUrl https://linotpserver/validate/simplecheck
 Timeout 5
 UserParam user
 PasswordParam pass
 UrlMethod GET
 BadPasswordKeyword ":-\("
 BadUserKeyword ":-/"
 AuthOKKeyword ":-\)"
 PasswordEncryption Clear
</AuthBy>
<Realm DEFAULT>
 AuthBy LinOTPURL
 AcctLogFileName ./detail
</Realm>
```

All users in the default realm will now be authenticated by LinOTP.

5.4 Enabling RADIUS authentication with rlm_perl

Note: In redundant enterprise environments we recommend using the rlm_linotp module.

Nevertheless, an easy alternative way to setup LinOTP with FreeRADIUS is to use the FreeRADIUS module `rlm_perl` and the LinOTP module `radius_linotp.pm`. Copy the file `radius_linotp.pm` to your preferred location. Then you need to configure `rlm_perl` as the FreeRADIUS module to be used, e.g. by setting it as the default authentication type in `/etc/freeradius/users`:

```
DEFAULT Auth-type := perl
```

Then you need to configure `rlm_perl` by telling it, where to find `radius_linotp.pm` by configuring `/etc/freeradius/modules/perl`:

[7] http://www.open.com.au/radiator/index.html
[8] http://www.open.com.au/

```
perl {
   module = /path/to/radius_linotp.pm
}
```

Now you need to configure the authentication of your site. In `/etc/freeradius/sites-enabled/linotp2`:

authenticate{ perl [....]

5.5 Enable SAML Identify Provider

Starting with 2.4 LinOTP integrates into simpleSAMLphp [9] as an Identity Provider. simpleSAMLphp is a pluggable solution for the SAML and shibboleth protocol. It can run as a Service Provider and as an Identity Provider. In the later case LinOTP provides a module to enable simpleSAMLphp to authenticate users with one time passwords against LinOTP. The LinOTP SAML module comes with the following files:

```
linotp2/
linotp2/linotp2.json
linotp2/docs/linotp2.txt
linotp2/lib/Auth/Source/linotp2.php
```

which need to be copied to the simpleSAMLphp modules directory. The authentication module is then addressed by `linotp2:linotp2`. See the file `docs/linotp2.txt` for configuring the module. For setting up simpleSAMLphp as a SAML Identity Provider you may want to refer to the simpleSAMLphp documentation [10].

[9] http://simplesamlphp.org/
[10] http://simplesamlphp.org/docs/1.8/simplesamlphp-idp.

6 Customization

LinOTP web components are based on templates and CSS. You may change the look and feel of the WebUI but we recommend not to change the default templates and style sheets, as an update may destroy your changes. Therefor you can define custom templates and style sheets.

6.1 Templates

In the file linotp.ini you can define a variable:

```
custom_templates = /etc/linotp2/custom-templates/
```

in the section `[app:main]`. All templates will be searched in this directory. If the template does not exist in this custom directory, the default template is used – usually from:

```
/usr/share/pyshared/linotp/templates.
```

So if you want to modify the look and feel, copy the template from the default location to this custom location and keep the structure of the sub directories. When you changed the custom templates you need to empty the Pylons cache:

```
rm /etc/linotp2/data/* -fr
/etc/init.d/apache2 restart
```

6.2 Style Sheets

The style sheets are located at `/usr/lib/pymodules/python2.6/linotp/public/`. The default styles sheets are located at:

```
linotp/public/manage/style.css
linotp/public/selfservice/style.css
linotp/public/openid/style.css
```

You can create the following files to overwrite the default style sheets. Creating these new files, will avoid the changes from being overwritten in case of a system update:

```
linotp/public/manage/custom-style.css
linotp/public/selfservice/custom-style.css
linotp/public/openid/custom-style.css
```

6.3 Changing the Logo

On the `/manage` and on the `/selfservice` portal you can change the logos easily. The LinOTP logo is defined as a background image of the div `#logo`. Take a look

at the style sheets `/usr/lib/pymodules/python2.6/linotp/public/manage/style.css` and `/usr/lib/pymodules/python2.6/linotp/public/selfservice/style.css`. There is the id definition `#logo` which points the the corresponding image file at `/usr/lib/pymodules/python2.6/linotp/public/images/`. Change the style.css to your custom-style.css and drop in a new image.

7 | Database connection

This section describes extended possibilities to configure the LinOTP token database.

7.1 Running LinOTP with DB2 on RedHat

This describes howto setup LinOTP with a connection to a DB2 database on a RedHat system. On other distributions this may vary.

7.1.1 Overview

You need to adapt the following config files:

- `/etc/linotp2/linotp.ini`
- `/etc/odbc.ini`
- `/etc/odbcinst.ini`
- `db2cli.ini` or `db2dsdriver.cfg`

You need to install the following packages:

- dsdriver (out of the scope of this document)
- unixODBC
- ibm_db_sa_linotp (shipped by LSE)
- pyODBC
- LinOTP (shipped by LSE)

7.1.2 Installation

Install unixODBC

You need to install unixODBC and the devel package as some parts of the python module pyODBC need to get compiled:

```
yum install unixODBC unixODBC-devel gcc-c++
```

Install pyODBC

Given you installed LinOTP to a virtualenv like `/opt/LinOTP`, you now can install the ODBC python module:

```
cd /opt/LINOTP
source bin/activate
pip install pyodbc
```

Install ibm_db_sa

This is the extension for SQLAlchemy, so that sqlalchemy is able to talk to DB2. You received this package from LSE. To install in within the virtualenv issue the command:

```
pip install /path/to/package/ibm_db_sa_linotp-0.2.2.tar.gz
```

Install LinOTP

Starting with version 2.5 LinOTP runs nicely with DB2. So install the latest LinOTP package:

```
pip install --upgrade /path/to/package/LinOTP-2.5.0.tar.gz
```

7.1.3 Configuration

LinOTP

Adapt the config file `/etc/linotp2/linotp.ini`:

```
sqlalchemy.url = ibm_db_sa+pyodbc://linotp_db
```

"linotp_db" is an identifier you may choose to your flavor. This identifier references the configuration in the following config files.

If you want to use the SQLAudit module, you can add a corresponding line accordingly. If the audit data is supposed to be located in another database, you would need to choose a second identifier like "audit_db".

ODBC

You need to tell ODBC that it is also capable of talking to DB2 databases. You can do this by adding the following to the file `/etc/odbcinst.ini`:

```
[DB2]
Description = DB2 ODBC driver
Driver      = /usr/local/dsdriver/lib/libdb2.so
```

Now you can define a new database connection by adding an entry to the file `/etc/odbc.ini` with the corresponding identifier from the `linotp.ini` file:

```
[linotp_db]
Driver=DB2
Description=My LinOTP Database connection
```

DB2

Now you can define the connection to the database by editing the file `db2dsdriver.cfg`. This file can be located in a system wide directory or in the home directory of the linotp user which runs the LinOTP wsgi process.

The file might look like this:

```
<configuration>
   <dsncollection>
      <dsn alias="linotp_db" name="lotp22" host="172.16.200.148" port="50000">
         <parameter name="CommProtocol" value="TCPIP"/>
         <parameter name="LobAsLongDataMode" value="1" />
         <parameter name="UserID" value="db2inst4" />
         <parameter name="Password" value="test123!" />
      </dsn>
   </dsncollection>
</configuration>
```

Note: The *name* and the *alias* in the DSN configuration should be different. Otherwise you may experience problems (missing password) when connecting to the database.

Within this file you can also define a redundant database access.

To test the connection run:

```
isql linotp_db
```

Note: If you defined the connection in the home directory of the linotp user, then you need to run the command as the linotp user.

Note: For more information on the `db2bsdriver.cfg` file see the IBM documentation like [1].

Create the database tables

Note: Please assure the following prerequisites:

- The database and the user for this database who has the right to create tables are already created.
- The database needs to be created with codepage “UTF-8”.
- You also need to create a temporary table space.
- The minimum pagesize needs to be 16384.

Now you can create the database tables by issuing the command:

```
paster setup-app /etc/linotp2/linotp.ini
```

7.2 Running LinOTP with Oracle on RedHat

This describes howto setup LinOTP with a connection to an Oracle database on a RedHat system. On other distributions this may vary.

7.2.1 Overview

You need to install the following packages:

- Oracle client (out of the scope of this document)
- cx_Oracle python module

[1] http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.swg.im.dbclient.config.doc/doc/r0054557.html

- LinOTP

Therefore you need to change the following config files:

- `/etc/linotp2/linotp.ini`

7.2.2 Installation

Install cx_Oracle python module

Change to your virtualenv and install cx_Oracle [2] to your virtualenv:

```
pip install cx_Oracle
```

Install LinOTP

Starting with version 2.4.3-4 LinOTP runs nicely with Oracle. So install the latest LinOTP package:

```
pip install --upgrade /path/to/package/LinOTP-2.4.3-4.tar.gz
```

7.2.3 Configuration

LinOTP

Adapt the config file `/etc/linotp2/linotp.ini`:

```
sqlalchemy.url = oracle://user:pass@host:port/dbname
```

Create the database tables

The database tables are created in the usual way:

```
paster setup-app /etc/linotp2/linotp.ini
```

7.3 Running LinOTP with Microsoft SQL

This describes howto setup LinOTP with a connection to a Microsoft SQL Server. In this example LinOTP is running on a RedHat system using the pip install method. On other distributions this may vary.

7.3.1 Overview

You need to adapt the following config files:

- `/etc/linotp2/linotp.ini`
- `/etc/odbc.ini`
- `/etc/odbcinst.ini`
- `/usr/local/etc/freetds.conf`

You need to install the following packages:

- unixODBC

[2] You can also download and install cx_Oracle from http://cx-oracle.sourceforge.net/ for your given Python version.

- pyODBC
- LinOTP (shipped by LSE)
- FreeTDS [3] version 0.91

7.3.2 Installation

Install unixODBC

You need to install unixODBC and the devel package as some parts of the python module pyODBC need to get compiled:

```
yum install unixODBC unixODBC-devel gcc-c++
```

Install pyODBC

Given you installed LinOTP to a virtualenv like `/opt/LinOTP`, you now can install the ODBC python module:

```
cd /opt/LINOTP
source bin/activate
pip install pyodbc
```

Install FreeTDS

You need to download version 0.91 of FreeTDS from [1] and build it.

This can usually be done running:

```
./configure
make
make install
```

The database driver gets installed to `/usr/local`. If you want to use any other directory, you need to specify it as parameter for the configure script.

7.3.3 Configuration

LinOTP

Adapt the config file `/etc/linotp2/linotp.ini`:

```
sqlalchemy.url = mssql+pyodbc://username:password@linotpdb
```

"linotpdb" is an identifier you may choose to your flavor. This identifier references the configuration in the following config files.

If you want to use the SQLAudit module, you can add a corresponding line accordingly. If the audit data is supposed to be located in another database, you would need to choose a second identifier like "audit_db".

[3] http://freetds.schemamania.org/

ODBC

You need to tell ODBC that it is also capable of talking to MS SQL databases. You can do this by adding the following to the file `/etc/odbcinst.ini`:

```
[FreeTDS]
Description=FreeTDS driver
Driver=/usr/local/lib/libtdsodbc.so
```

Now you can define a new database connection by adding an entry to the file `/etc/odbc.ini` with the corresponding identifier from the `linotp.ini` file:

```
[linotpdb]
# see http://www.freetds.org/userguide/odbcconnattr.htm
Driver=FreeTDS
Description=LinOTP database
Servername=TDSServer
Port=1433
Database=linotp2
ClientCharset=UTF-8
tds_version = 7.0
```

FreeTDS

Now you can define the connection to the database by editing the file `freetds.conf`.

This file usually is located at `/usr/local/etc`. You should add a section like this:

```
[TDSServer]
host = 172.16.200.148
port = 1433
instance = SQLEXPRESS
tds version = 7.0
client charset=UTF-8
```

There you can specify the hostname of the Database server, the port and the instance you want to use.

To test the connection run:

```
isql linotpdb username password
```

Create the database tables

Note: Please assure the following prerequisites:

- The database and the user for this database who has the right to create tables are already created.

Now you can create the database tables by issuing the command:

```
paster setup-app /etc/linotp2/linotp.ini
```

7.4 Special parameters for database connections

This section describes some special parameters for the database connection.

7.4.1 Pool recyle

This is a special parameter for the SQL Audit module.

After this certain amount of time (in seconds) connections from the database connection pool will not be used anymore:

```
linotpAudit.sql.pool_recyle = 3600
```

7.4.2 Implicit returning

PostgreSQL, Oracle and MS SQL support RETURNING [4].

`implicit_returning` indicates that RETURNING can be used by default to fetch newly inserted primary key values. [5] In certain scenarios like redundant MS SQL servers it is reported that this causes problems. In the `linotp.ini` this can be turned off by adding the config:

```
linotpSQL.implicit_returning = False
```

[4] http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/returninginto_clause.htm
[5] http://docs.sqlalchemy.org/en/latest/core/schema.html

8 | Security Modules

Starting with version 2.5 LinOTP supports different security modules.

The security module specifies how the encryption of the HMAC keys and password is handled in LinOTP.

The default behaviour having 3 AES encryption keys in a file */etc/linotp2/encKey* is also seen as a security module.

8.1 Defining Security Modules

You can define several different security modules. At the moment LinOTP comes with the modules

`Pkcs11SecurityModule` which is defined in the python module `linotpee.lib.security.pkcs11`

`DefaultSecurityModule` which is defined in the python module `linotp.lib.security.default`.

But this can easily enhanced to write your own security module.

To define modules you follow this scheme in the */etc/linotp2/linotp.ini* file:

```
linotpSecurity.some_name_1.module = linotpee.lib.security.some_package_1.some_class_1
linotpSecurity.some_name_2.module = linotpee.lib.security.some_package_2.some_class_2
linotpSecurity.some_name_3.module = linotp.lib.security.some_package_3.some_class_3
linotpSecurity.some_name_4.module = linotp.lib.security.some_package_4.some_class_4
```

You then define, which security module should be used by LinOTP. This is also done in the *linotp.ini* file:

```
linotpActiveSecurityModule = some_name_2
```

Note: If you define nothing in the *linotp.ini* LinOTP uses the default security module and will work as usual.

8.2 Defining SafeNet LunaSA

Note: Please also refer to the original Luna SA HTML documentation for more detailed information!

LinOTP is well tested with the SafeNet LunaSA 4.5 PED.

To have the encryption handled by the LunaSA a PKCS11 security module is used.

You need to define it in the *linotp.ini* like this:

```
linotpSecurity.lunasa.module = linotpee.lib.security.pkcs11.Pkcs11SecurityModule

linotpActiveSecurityModule = lunasa

linotpSecurity.lunasa.library = libCryptoki2_64.so
```

```
linotpSecurity.lunasa.configLabel =config
linotpSecurity.lunasa.valueLabel =value
linotpSecurity.lunasa.tokenLabel =token
linotpSecurity.lunasa.defaultLabel =token
linotpSecurity.lunasa.password = /RMF-At5F-p6XJ-HR64
linotpSecurity.lunasa.slotid = 1
```

The parameters “library”, “configLabel”, “valueLabel”, “tokenLabel”, “defaultLabel”, “password” and “slotid” are module dependent parameters. Other future modules may have other parameters.

`library` This is the PKCS11 library. In case of the LunaSA this is the *libCryptoki2_4.so.*

`password` The password of the PKCS11 slot aka. the smartcard PIN.

`slotid` This is the slot where the AES keys are located. In case of the LunaSA this is the partition. You can check for the slot number by issuing the command `vtl verify`.

`configHandle`, `valueHandle`, `tokenHandle`, `defaultHandle` These are the handles of the AES key objects in the slot. *configHandle*, *valueHandle* and *tokenHandle* are used. If on of the parameters is missing, the *defaultHandle* is used.

`configLabel`, `valueLabel`, `tokenLabel`, `defaultLabel` These are the labels of the AES Key objects in the slot.

Note: If you are running a LunaSA HA group you must use the labels. The handles will not work.

`password` Here you can enter the password of the partition. If the password is not specified, you can pass it to LinOTP later.

Note: Instead of using the name “lunasa” you can use any other specifier.

8.2.1 Partition Password

If you do not want to store the password in the *linotp.ini* you can pass this to LinOTP manually.

After starting the LinOTP service you can check the security module like this:

```
linotpadm.py -U https://localhost -a admin -C securitymodule
```

You will get the following response:

```
{   u'status': True,
 u'value': {   u'setupSecurityModule': {   u'activeSecurityModule': u'lunasa',
                                           u'connected': True}}}
```

If `connected` is `False` then you need to provide the partition password for the HSM slot:

```
linotpadm.py -U https://localhost -a admin -C securitymodule --module=lunasa
```

Now you will be asked for the password of the partition.

8.3 Setting up SafeNet LunaSA

Warning: This documentation does not replace the SafeNet LunaSA documentation. The HSM is a sophisticated device - you should consult the manual and know what you are doing.

8.3.1 Requirements

You need to install the following software packages on the LinOTP server that were delivered with your HSM:

- ctp-4.5.0
- libcryptoki-4.5.0
- vtl-4.5.0

The components are installed to */usr/lunasa*. The executables are located at */usr/lunasa/bin*.

8.3.2 Network settings

Note: For connecting to the Luna SA you need to connect the Luna SA appliance with the client computer via a null modem cable with the following settings: Serial port baud rate: 115200 N,8,1 (no parity, 8 data-bits, one stop-bit) VT-100 terminal emulation. Hardware flow

---

Alternatively the HSM is accessible via IP 192.168.0.1. After the first login with the username `admin` and the password `chrysalis` the password is requested to be changed. Furthermore the time needs to be set and the network should be configured:

```
# setting time zone
lunash:> sysconf timezone -set Europe/Berlin
# setting time
lunash:> sysconf -time 12:55 20071223
# setting hostname
lunash:> net hostname hsm1
# set domain name
lunash:> net domain example.com
# set multiple nameservers
lunash:> net dns -nameserver 172.16.16.6
lunash:> net dns -nameserver 172.16.16.7
# set multiple search domains
lunash:> net dns -search example.com
# set eth0. (eth1 may also be set)
   lunash:> net interface -static -device eth0 \
   -ip 172.16.16.102 -netmask 255.255.255.224 \
   -gateway 172.16.1
# control the settings
lunash:> net show
```

Now the LunaSA can be contacted via ssh. When the network connection is working correctly an ntp service can be set up. Setting up the domain controller in forest root as NTP servers:

```
lunash:> sysconf ntp -addserver 172.16.16.6
```

8.3.3 LunaSA server certificate

Note: For communication the LunaSA generates a certificate. For correct generation the LunaSA needs to be inserted in the DNS servers or in /etc/hosts.

When the DNS server resolve the hsm1 correctly the server certificate can be generated:

```
lunash:> sysconf regenCert
CAUTION: Current Server Certificate and Private Key will be
overwritten. All clients will have to add the server
again with new certificate.
```

```
Type 'proceed' to generate cert or 'quit' to cancel
> proceed
```

To be able to use the LunaSA via network, the trusted interface has to be defined:

```
lunash:>ntls bind eth0
```

8.3.4 Initialization of HSM

To be able to initialize the HSM the Luna PED needs to be connected to the LunaSA appliance and you need to got a set of PED Keys. The LunaSA is configured via the `hsm init` command. Most of the parameters for this command are entered via the Luna PED.:

```
lunash:> hsm init -label hsm1
```

Note: You should stick to the web based documentation closely, since this is a sensitive process.

Roughly after having issued the hsm init command the process is as follows:

Create HSM Admin PED Key

- Insert the blue PED key. This will be the ''HSM Admin PED Key''.
- As the fresh key is blank, a new PED PIN needs to be chosen.
- by `Copy this PED Key` backup copies of the PED key can be generated.
- Login as HSM Admin (Security Officer /SO).

Create Domain PED Key

- Insert a second PED key. This will be the ''Domain PED Key''.
- If this is a fresh key, a new PED PIN should also be set.
- Backups can be generated.

The initialization of the HSM has finished now. Copies of the PED Keys can also be made later.

HSM security polices

Using the command:

```
hsm showPol -c
```

you can display the policies:

```
Description                          Value       Code      Destructive
===========                          =====       ====      ===========
Allow cloning                        On          7         Yes
Allow non-FIPS algorithms            On          12        Yes
Allow MofN auto-activation           On          13        No
SO can reset partition PIN           On          15        Yes
Allow network replication            On          16        No
Allow Remote Authentication          On          20        Yes
Force user PIN change after set/reset Off        21        No
```

For performing Backups the policy *Allow cloning* must be ON. For a redundant HA setup the policies *Allow cloning* and *Allow network replication* must be ON.

To switch a policy to ON use the command:

```
hsm changePol -p 7 -v 1
```

Create HSM Partitions

The LunaSA HSM can be partitioned that way, that each LinOTP is using an own partition of the HSM. To create a new partition on the HSM you must connect the Luna PED and logon as HSM admin issuing the command:

```
lunash:> hsm login
```

and inserting the blue HSM Admin PED Key.

A new partition is created issuing the command:

```
lunash:> partition create -name yourPartition
```

A black Partition Owner PED Key is generated. A PIN for the black PED Key needs to be set. When asked `Are you duplicating this PED Key Y/N?` backups of the black PED Key may be generated.

The Luna PED will now display the Password that clients (the LinOTP server) will use to authenticate to this partition. As this password will never show again anywhere else, it needs to be recorded/remembered:

```
Login secret value
btqx-EFGH-3456-7/K9
Please write it down.
(Press ENTER)
```

After displaying the client password the creation of the partition has finished.

If you have more partitions, create all other partitions with new black partition owner keys.

For each partition a separate black Partition Owner PED Key should be used. Otherwise the LunaSA will create a so called Group PED Key.

Note: When creating Group PED Keys the access rights to the HSM of the LinOTP servers can not be separated! It is recommended to use a separate PED Key for each partition.

Partition policies

Partition policies can be viewed on the Luna SA using the command:

```
lunash:> partition showPolicies -partition yourPartition
```

Activate Partitions

In order for an application to access the partition without the black partition owner key plugged in, the Partition needs to be activated. Therefor the Policy `Allow activation` needs to be set to 1:

```
lunash:> partition changePolicy -partition
         yourPartition -policy 22 -value 1
```

For setting the partition policy you need to have the blue SO PED key. Afterwards the partition can be activated:

```
lunash:> partition activate -partition partitionPolicyCA
```

When activating the partition you need to enter the client password that was generated when the partition was initialized. For activating the partition you need to have the Partition Owner PED key.

If the HSM lost power and you start the HSM again, the partition needs to be activated again. To avoid this, you can turn the `Autoactivation` policy on:

```
lunash:> partition changePolicy -partition
         yourPartition -policy 23 -value 1
```

8.3.5 Setting up HSM clients and assigning clients to HSM partitions

A LinOTP server talking to the HSM is called a HSM client. The connection is encrypted and authenticated via certifcates on both sides. The certificate of the LunaSA was already generated. This server certificate needs to be transferred to each LinOTP server.

Copy the server certificate to each LinOTP by issuing the command:

```
./ctp admin@hsm1:server.pem .
```

You need to add the HSM server on the client side:

```
./vtl addServer -n hsm1 -c server.pem
```

Now the client needs to get a client certificate created:

```
./vtl createCert -n linotp
```

Copy the client certificate to the LunaSA:

```
./ctp cert/client/linotp.pem admin@hsm1:
```

Now the client needs to be registered on the LunaSA and be assigned to a partition. Therefore on the LunaSA the admin must issue the following commands:

```
# register the client
lunash:> client register -client linotp -hostname linotp
# assign a client to partition
lunash:> client assignPartition -client linotp -partition yourPartition
```

Verify the working connection by:

```
./vtl verify
```

You should see a list with the available slots. You also need the slot number to configure later in LinOTP.

8.3.6 Troubleshooting

The names must resolve successfully. Try to ping the HSM from the LinOTP server by name and the LinOTP server from the HSM:

```
lunash:> net ping linotp
```

It could be that the ntls service needs to be restarted:

```
lunash:> service restart ntls
```

8.4 Create AES Keys

You can create AES keys on the HSM using the security module:

```
python linotpee/lib/security/pkcs11.py
```

8.5 Backup and restore with LunaSA

You need to backup the contents of the partition you are using.

With the LunaSA you need a backup token, which in turn is an HSM itself. This is why you need the following cloning policies enabled:

Cloning HSM:

```
hsm changePol -p 7 -v1
```

Secret Key Cloning per partition:

```
partition changePol -pa yourPartition -po 4 -v1
```

8.5.1 Backup

Insert the Backup-Token into the lower slot.

Issue the command:

```
hsm login
```

Then start the backup procedure:

```
partition backup -partition HSMPartitionname -password ClientPassword
```

Note: During the backup procedure the backup token (which in turn is an HSM) gets its own blue and black key. But you can also use the same keys you where using for the original partition on the HSM. The same red Domain-Key **must** be used.

Note: You can only backup **one** partition to a backup token.

Issuing the backup command will result in the following output:

```
CAUTION:  Are you sure you wish to initialize the backup
       token named:
         no label
       Type 'proceed' to continue, or 'quit' to quit now.
       > proceed
Luna PED operation required to initialize backup token - use Security Officer (blue) PED key.

Luna PED operation required to login to backup token - use Security Officer (blue) PED key.

Luna PED operation required to generate cloning domain on backup token - use Domain (red) PED key.

Luna PED operation required to generate partition backup space on token - use User or Partition Owner (black) P

Luna PED operation required to login to partition backup space on token - use User or Partition Owner (black) P
```

Then the objects that are being backupped are listed.

Warning: During the backup process the handles of the keys may change. So you should also memorize and record the labels of the keys, since during restore the keys might get restored to other handles!

8.5.2 Restore

In case of recovering a broken HSM or in case of setting up a HA solution, you need to restore the data. You need to login to the HSM and create a new partition with the same name as the old one:

```
hsm login

partition create -par yourOldPartition
```

Note: You can reuse the existing black partition owner key

Again you need to set the partition policies:

```
partition changePolicy -par testCA -pol 22 -v 1
partition changePolicy -par testCA -pol 23 -v 1
```

Finally insert the backup token into the lower slot of the HSM and start the restore process:

```
partition restore -par yourOldPartition -password /RMF-At5F-p6XJ-HR64 -replace

       > proceed
Luna PED operation required to login to partition backup space on token - use User or Partition Owner (black) P

Luna PED operation required to activate partition on HSM - use User or Partition Owner (black) PED key.
```

Note: If this is a new machine, then of course you need to setup the trust link with the clients anew.

Note: In case you need to recover a failed member, use the command `haadmin -recover`.

8.6 Setting up HA and Load balancing for LunaSA

Several LunaSAs can join a HA group.

You have to configure the LinOTP machine with each LunaSA. Using the vtl tool you can now create the HA group.

Note: The policies “Allow Cloning” and “Allow Network Replication” must be turned on. Use “hsm setPolicy” to set those policies if necessary.

Note: Both partitions need to have the “AutoActivated” policy turned on.

Note: Both HSMs need to use the same red Domain key.

Note: The HA is set up between two partitions on two HSMs. Therefore these partitions need to have the same password. The partitions do not need to have the same name.

Change the passwords of the partitions, so that the partitions have the same password:

```
partition changePw
```

Use “partition show” to record the serial number of the partition.

8.6.1 Register LinOTP

You need to register the LinOTP (client) with both HSMs as described in section *Setting up HSM clients and assigning clients to HSM partitions* (page 152).

8.6.2 Creating HA group

Finally when the `vtl verify` command shows you both HSMs you can setup the HA group:

```
./vtl haAdmin -newGroup
              -serialNum <serialnumber-of-first-HSM>
              -label <label-of-HA-group>
              -password <partition-Password>
```

The file `/etc/Chrystoki.conf` now should have a new entry `VirtualToken`.

Note: Internally the partition gets an HA key created to identify to which HA group this partition belongs. If this new HA group is a copy of a group on another LinOTP server, you will be warned that there is an existing HA key on this partition. If you want to have both LinOTP servers talk to this same HA group, you must type ‘copy’ to keep the existing HA key. If you want to start over with the HA group, then type ‘remove’. The HA key on this partition will be removed.

For adding further members to the group, you need the HA group number. You can either see this number in the `cryptoki.conf` file or you can see this number by issuing the command:

```
./vtl haAdmin -listGroups
```

You can now add the second HSM to the HA group:

```
./vtl haAdmin -addMember
              -group <serialnumber-of-the-ha-group>
              -serialNum <serialnumber-of-second-HSM>
              -password <partition-password>
```

Finally, when all members are added, you need to issue the command:

```
./vtl haAdmin -synchronize -group <group-label>
```

Note: In case you need to recover a failed member, use the command `haadmin -recover`. For more details see section *Restore an HA group* (page 156).

The “VirtualToken” in the file `cryptoki.conf` now should contain both the serial numbers of the two partitions.

Note: The `vtl verify` command will not show the virtual token. You can use the `cmu list` tool to list all three slots. The virtual token (HA) usually will be slot #3. Using `cmu list` you should also list the objects in the virtual slot to check, which handles the three encryption keys were assigned.

Please reconfigure `/etc/linotp2/linotp.ini` to use the “HA Virtual Card Slot”.

8.6.3 Monitoring

On the LinOTP machine you can use the command:

```
/usr/lunasa/bin/vtl haAdmin -status -show
```

to check which HSM is alive.

Restore an HA group

Usually you will not have to restore using the backup token.

If only one member of the HA group failed, you can use the command:

```
./vtl haAdmin -recover <group name>
```

which will recover a failed member (power outage) to the HA group.

If you had a hardware failure and need to install a new HSM, you need to remove the broken member from the HA group and add the new HSM to the HA group:

1. Remove the broken HSM from the HA group using the command:

   ```
   vtl haAdmin -removeMember <group-name> -serialNum <serial-of-the-failing-partition>
   ```

2. Initialize the new HSM, create the new partition, assign the partition to the client, set the partition password and
3. add the partition to the HA group using the command `vtl haAdmin --addMember`.
4. Synchronize the HA group, so that the keys are synchronized to the new HSM:

   ```
   vtl haAdmin -synchronize -group <group-name>
   ```

If both of your HSMs fail, you need to setup both HSMs with the HA groups from the scratch (see *Restore* (page 154)). Then you need your backup token to populate the first partition with the keys again.

Note: In this case, the handles of the keys may have changed. Check if you need to adapt `/etc/linotp2/linotp.ini`.

8.7 Managing Passwords with LunaSA

To change existing passwords you can do the following.

Note: In nearly all cases when changing passwords you need physical access to the HSM via the PED keys!

8.7.1 Changing admin Password

To change the password of the admin user you can issue the command:

```
lunash:> user password admin
```

8.7.2 Changing HSM PED Password

You can change the PIN of the blue PED key using the command:

```
lunash:> hsm changePw
```

Warning: Blue HSM admin key needed!

8.7.3 Changing Partition Passwords

You can change the partition password and the PIN of the black partition owner key:

```
lunash:> partition changePw -partition <partition-name>
```

Warning: Black partition owner key needed!

8.7.4 Resetting Partition Passwords

If you have forgotten the partition password you can reset the partition password as HSM admin:

```
lunash:>hsm login

lunash:>partition resetPw -partition <partition-name>
```

Warning: Blue HSM admin key needed!

9 Integration examples

9.1 OTP Authentication with Apache2

There are two possible RADIUS authentication modules for Apache2. `mod_auth_radius` is packed with the FreeRADIUS server. `mod_auth_xradius` can be downloaded from http://outoforder.cc.

Previous versions of `mod_auth_radius` had the problem to not work with OTPs, but this is now fixed in the current version.

This describes the use with `mod_auth_xradius`.

Note: If you do not want to use RADIUS, you can also use the module `mod_authn_linotp`, which is available on github.

9.1.1 Download

Go to http://www.outoforder.cc/projects and download `apr_memcache` and `mod_auth_xradius`. You will also need to install `apache2-prefork-dev` , `apache2-utils` , `libmemcache-dev and libapreq2- dev` [1]

9.1.2 Compile

First you need to compile `apr_memcache`. The authentication module uses the memcache client to talk to the memcached daemon.

In the `apr_memcache` source directory a:

```
./configure prefix=/usr
make && make install
```

should do.

Now you can compile `mod_auth_xradius`:

```
./configure --with-apxs=/usr/bin/apxs2
make && make install
```

9.1.3 Configuration

Now you need to assure, that the memcached daemon is started at system boot time. There are different ways depending on your distribution.

Your Apache2 configuration might look like this:

[1] These are Debian package names. On other distributions the names may vary.

```
LoadModule auth_xradius_module /usr/lib/apache2/modules/mod_auth_xradius.so
AuthXRadiusCache memcache "127.0.0.1:11211"
## 10h Timeout in seconds.
AuthXRadiusCacheTimeout 36000
<VirtualHost *>
  ServerName secret-site3
  ServerAdmin webmaster@localhost
  DocumentRoot /var/www/secure
  <Directory /var/www/secure/>
     Options Indexes FollowSymLinks MultiViews
     AllowOverride None
     Order allow,deny
     allow from all
     AuthType Basic
     require valid-user
     AuthBasicProvider xradius
     AuthName "Please enter your username and OTP-Password and OTP-PIN"
     AuthXRadiusAddServer "localhost:1812" "testing123"
     AuthXRadiusTimeout 7
     AuthXRadiusRetries 2
  </Directory>
  ErrorLog /var/log/apache2/error.log
  LogLevel debug
  CustomLog /var/log/apache2/access.log combined
  ServerSignature On
</VirtualHost>
```

9.2 Firewall integration

One common scenario is to use one time passwords for remote access via SSL VPNs. In case of the Astaro Security Gateway it could look like this.

Within Users → Authentication → Servers you need to add your LinOTP host. (see *Configure Astaro Security Gateway. Add LinOTP host.* (page 161))

Choose “RADIUS” for the new server and enter the shared secret that was configured on the RADIUS server on your LinOTP machine. (see *Configure Astaro Security Gateway. Enter RADIUS shared secret.* (page 162))

You may now test your setup by entering a valid username and its corresponding OTP PIN and OTP value.

You are now ready to use this setup to bind the authentication to the users. (see *Configure Astaro Security Gateway. Test your setup.* (page 163))

Note: With the FreeRADIUS module of LinOTP 2.2 mschapv2 is not supported. Thus OTP can only be used for SSL VPN but not for PPTP.

9.3 Authentication with third party OTP solutions

If you need to authenticate users with existing tokens, that are not supported by LinOTP you can setup the FreeRADIUS server to decide, if a user is authenticated against LinOTP or the other solution like RSA SecurID(R) or other unsupported token systems.

A common scenario is to have the FreeRADIUS server check, if a user is a member of a certain LDAP [2] group.

[2] Of course, this also works for Active Directory.



Figure 9.1: **Configure Astaro Security Gateway. Add LinOTP host.**



Figure 9.2: **Configure Astaro Security Gateway. Enter RADIUS shared secret.**

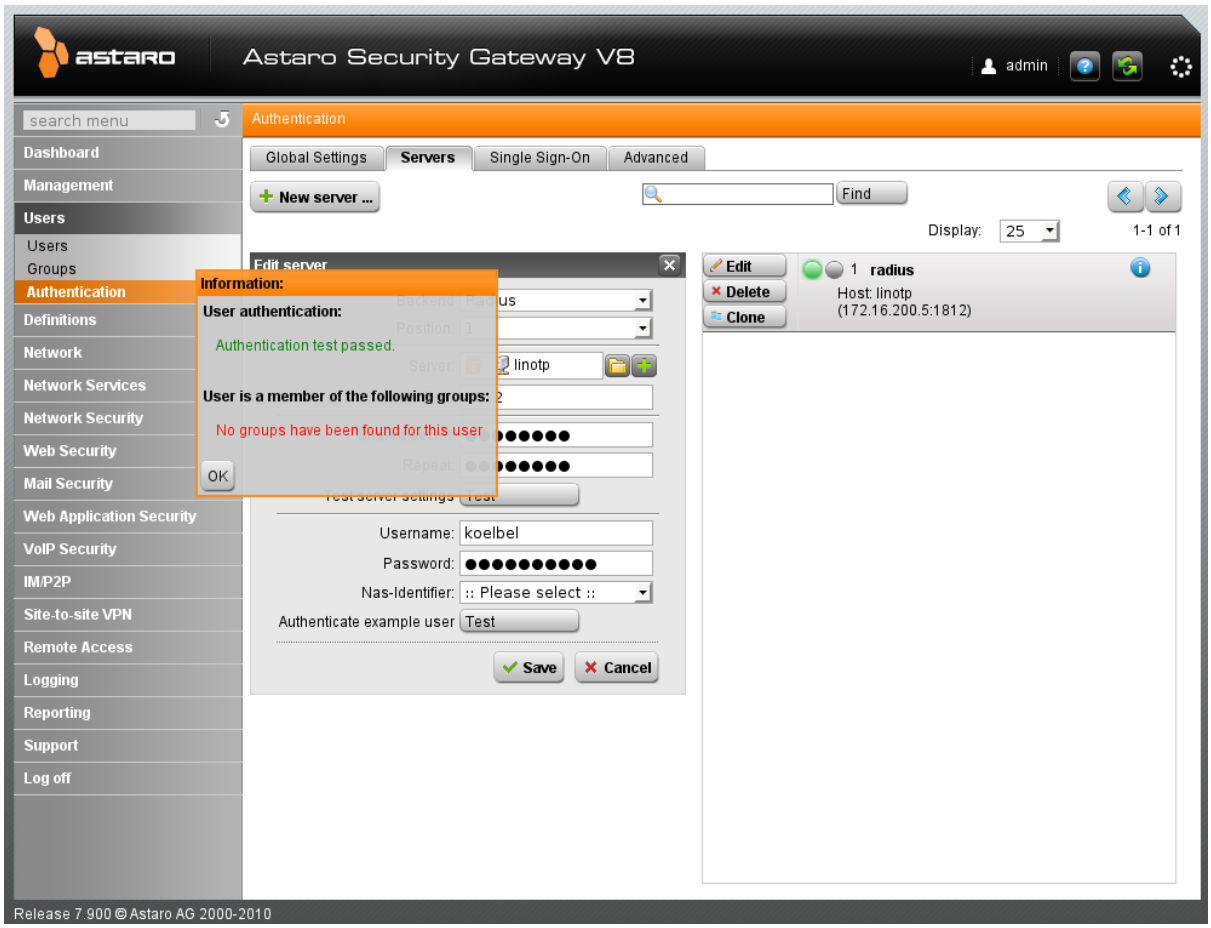


Figure 9.3: **Configure Astaro Security Gateway. Test your setup.**

By putting users in certain LDAP groups, you can define which backend system should authenticate the user:

If the user is either a member of this group you could have the request handled by LinOTP. If the user is not a member of this group the authentication request can be forwarded to another RADIUS server.

In the below example we will use the reverse logic and forward the authentication request to another 3rd party RADIUS server, *if* the user is a member of this group.

9.3.1 Configure FreeRADIUS LDAP

First you need to define the LDAP connection of your FreeRADIUS server.

Edit the file `/etc/freeradius/modules/ldap` like this:

```
ldap instance1 {
     #
     #  Note that this needs to match the name in the LDAP
     #  server certificate, if you're using ldaps.
     server = "172.16.200.61"
     # The DN of the user you are using to authenticate
     identity = "cn=admin,ou=linotp,dc=linotp-test,dc=local"
     password = your_password
     # BaseDN to search the users
     basedn = "ou=linotp2,dc=linotp-test,dc=local"
     filter = "(sAMAccountName=%{%{Stripped-User-Name}:-%{User-Name}})"
     #  How many connections to keep open to the LDAP server.
     #  This saves time over opening a new LDAP socket for
     #  every authentication request.
     ldap_connections_number = 5
     # seconds to wait for LDAP query to finish. default: 20
     timeout = 4
     #  seconds LDAP server has to process the query (server-side
     #  time limit). default: 20
     #
     #  LDAP_OPT_TIMELIMIT is set to this value.
     timelimit = 3
     #
     #  seconds to wait for response of the server. (network
     #   failures) default: 10
     #
     #  LDAP_OPT_NETWORK_TIMEOUT is set to this value.
     net_timeout = 1
     tls {
             start_tls = no
     }
     dictionary_mapping = ${confdir}/ldap.attrmap
     edir_account_policy_check = no
     groupmembership_filter = "(&(objectClass=group)(member=%{control:Ldap-UserDn}))"
}
```

You can define many different ldap instances by repeating the `ldap` block multiple times.

9.3.2 Configure FreeRADIUS proxy

In a RADIUS sense the third party server is a RADIUS proxy. Thus we need to define such a proxy in the file `/etc/freeradius/proxy.conf`:

```
realm FARWARDSERVER {
     authhost = 3rd.party.server:1812
     accthost = 3rd.party.server:1813
```

```
    secret   = radius-secret
}
```

LinOTP's FreeRADIUS needs to be configured as a RADIUS client on the 3rd party server. The 3rd party server and the RADIUS secret need to be configured in the above config file.

9.3.3 Configure proxy decision

Finally we need to confiugre the decision in which case the request is handled by LinOTP and in which case the request will be handled by the 3rd party server. This is done by adapting the `authorize` section in `/etc/freeradius/sites-enabled/linotp`:

```
authorize {
   [...]
   #
   # put the ldap instance here
   #
   instance1

   if ( instance1-Ldap-Group == "cn=3rdpartyusers,ou=linotp,dc=linotp-test,dc=local" ) {
     update control {
         Proxy-To-Realm = "FORWARDSERVER"
     }
   }
}
```

Then all users that are member of the group **3rdpartyusers** will be forwarded to the 3rd party servers and all other users will be authenticated by LinOTP.

If you need to define more complex conditions take a look at the FreeRADIUS unlang [3].

[3] http://freeradius.org/radiusd/man/unlang.html

10 | Updates

Warning: Before updating, please assure, that you have a backup of your encryption key and also of your database. Having a backup of your `linotp.ini` file is also a good idea!

10.1 Updating from LinOTP 2.5.1 to LinOTP 2.5.2

10.1.1 Updating a deb install

With version 2.5.2 the naming of some packages changed:

old name in version 2.5.1	new name in version 2.5.2
linotpuseridresolver	linotp-useridresolver
linotpuseridresolveree	linotp-useridresolver-ee
linotpdoc	linotp-doc
smsprovider	linotp-smsprovider

Transition packages with the old names are used to perform the update.

You need to issue the following command:

```
dpkg -i linotpuseridresolver_2.5.2-1_all.deb \
        linotpuseridresolveree_2.5.2-1_all.deb \
        linotpdoc_2.5.2-1_all.deb \
        smsprovider_2.5.2-1_all.deb \
        linotp_2.5.2-1_all.deb \
        linotp-ee_2.5.2-1_all.deb \
        linotp-useridresolver_2.5.2-1_all.deb \
        linotp-useridresolver-ee_2.5.2-1_all.deb \
        linotp-doc_2.5.2-1_all.deb \
        linotp-smsprovider_2.5.2-1_all.deb
```

Afterwards you can remove the old packages:

```
dpkg -r linotpdoc linotpuseridresolver linotpuseridresolveree smsprovider
```

10.1.2 Updating a pip install

Issue the following command to update your pip installation (*LinOTP Server Installation – the tar.gz, virtualenv and pip way* (page 109)):

```
pip install -upgrade linotp
```

Then upgrade the enterprise edition components. You need to download the latest version from the customer portal and issue the commands:

```
pip install --upgrade /path/to/LinOTP-EE-2.5.2.tar.gz
pip install --upgrade /path/to/LinOtpUserIdResolverEE-2.5.2.tar.gz
pip install --upgrade /path/to/LinOtpDoc-2.5.2.tar.gz
pip install --upgrade /path/to/Smsprovider-2.5.2.tar.gz
```

10.1.3 Changelog

Dokumentation

- Added documentation for MS SQL server support.
- Added howto for forwarding RADIUS request depending on LDAP group membership.
- Added Yubikey documentation for Yubikey NANO.

LinOTP Server

- Added dynamic token modules. All tokens can now be loaded dynamically.
- Added policy import and export.
- Added possibility to display action history in selfservice.
- Added new Token: Yubikey in orignial yubikey mode (44 characters) to authenticate with the yubico online cloud service.
- Added a script (linotp-pip-update) to update a pip installation.
- Added authentication to ocra controller.
- Added the possibility to give the CA certificate with the LDAP Resolver when using LDAPS.
- Added univention UCS / LinOTP documentation.
- Added users and resolvers to policies in selfservice, authentication, enrollment and authorization.
- Added a policy checker to the WebUI.
- Assign Token by OTP value in selfservice.
- Implemented additional API to to a get_serial_by_otp in selfservice.
- Improved policies: exclude clients.
- Improved PSKC import to import OCRA suite.
- Increase font size (style italic) to make it easier to assign a token to a user.
- Limit size of realm and resolver dialogs. If hundret resolvers or realms are defined, the dialog is too big.
- Make the cookie a secure cookie, means it must be transferred via SSL
- Performance fix - reduce userid lookup.
- Add poissibility to set maximum auth count and validity period.
- The mobile number (instead of phone) will now be used in selfservice for SMS token.
- closed: More detailed information when the SMS is sent via /validate/check of /validate/smspin.
- closed: The preset of the mobile number for an SMS token is now contained in the token.mako file.
- closed: The user was not able to authenticate to selfservice.
- closed: Deprecation Information about searching tokens.
- closed: Use SecureFormatter in linotp.ini.
- closed: The sms text from the policy is used to send the SMS.

- closed: We require python 2.6.
- closed: Make sure that genkey is in defined range.
- Renamed the webprovissionOCRA to activateQR.
- Reverted to the timeStepping=30 for the setup.
- fixed: Correct audit entry, when the userpassword (otppin=1) is wrong.
- fixed: Added a search button to flexigrid.
- fixed: Added SecureFormatter to be able to remove non printable characters from the log args
- fixed: The audit trail does not show entries with sqlalchemy 0.8.0
- fixed: The setting of the OCRA PIN does not work in the WebUI.
- fixed: Return space instead of empty string in case of MS SQL server
- fixed: Problems with redundant MS SQL server.
- fixed: Problem, that an admin was not able to view the users in the realm he has rights to.
- fixed: The broken FileAudit module.
- fixed: The possiblity to do cross site scripting in the doc controller. (serve documentation statically)
- fixed: Problems in token search.
- fixed: User enumeration with validate/smsping.
- fixed: Tokeniterator exact user match.
- fixed: Permissions for SSL privkey and who.ini.
- fixed: The system settings (WebUI) are not stored, if data on another tab is missing.
- fixed: OCRA bug for missing leading zeros - truncation to last digit.

GTK Client

- The Yubikey can now be enrolled with GTK client based on python 2.7.
- Modified the GTK client this way, that the realm filter is always available.
- Added the possibility to give the CA certificate with the LDAP Resolver.
- Added import of policies to GTK client.
- Added the possiblity to export the policies to a file.
- Audit log now shows the last entry first.
- Added eToken enrollment command line tools.
- Fixed missing dependency for configobj.
- Fixed the jumping of the filter cursor.
- Fixed display of policy in GTK client.

10.2 Updating from LinOTP 2.5.0 to LinOTP 2.5.1

10.2.1 Updating a deb install

Issue the command:

```
dpkg -i linotp_2.5.1_all.deb linotp-ee_2.5.1_all.deb linotpuseridresolveree_2.5.1_all.deb \
    linotpdoc_2.5.1_all.deb python-qrcode_2.4.2_all.deb
```

10.2.2 Updating a pip install

Warning: Before updating a pip installation you very much need to backup your files in /etc/linotp2! The pip installing logic is not that sophisticated, it might overwrite existing config files. So please backup at least: `/etc/linotp2/linotp.ini` and `/etc/linotp2/encKey`!

If you have installed LinOTP using pip as described in *LinOTP Server Installation – the tar.gz, virtualenv and pip way* (page 109), you first can upgrade the main server components via the internet to the latest version:

```
pip install -upgrade linotp
```

Then upgrade the enterprise edition components. You need to download the newer version the customers portal:

```
pip install qrcode
pip install --upgrade /path/to/packages/LinOTP-EE-2.5.1.tar.gz
pip install --upgrade /path/to/packages/LinOtpUserIdResolverEE-2.5.1.tar.gz
pip install --upgrade /path/to/packages/LinOtpDoc-2.5.1.tar.gz
```

10.2.3 Changelog

LinOTP Server

- added QR-Code enrollment in management web UI and selfservice portal
- added QR-Code image to reply
- added HTML documentation for LinOTP Web UI
- added import OCRA seeds via CSV
- added possibility to send 500er HTTP error instead of status:false
- added alert-box (pop under)
- added support for AD uidType DN, objectGUID and sAMAccountName
- added man pages for command line tools
- improved python PIP installation
- improved performance with dynamic token classes
- define the contents of the lost password token (#806)
- only active tokens are counted for the licensing (#810)
- using sqlalchemy for where clauses in SQLResolver
- fixed translation
- fixed broken totp resync
- fixed empty password are neglected ldap_simple bind
- fixed connection close() in checkMapping()

10.3 Updating from LinOTP 2.4.4 to LinOTP 2.5.0

10.3.1 Updating a deb install

Before updating, please assure, that you have a backup of your encryption key and also of your token database.

Issue the command:

```
dpkg -i linotp_2.5.0-8_all.deb linotp-ee_2.5.0_all.deb linotpuseridresolveree_2.5.0-2_all.deb
```

If you want to use OCRA functionality you also need to update your database. You can do this by issuing the command:

```
paster setup-app /etc/linotp2/linotp.ini
```

After this please check the access rights of your logfiles in `/var/log/linotp/`.

10.3.2 Updating a pip install

Warning: Before updating a pip installation you very much need to backup your files in /etc/linotp2! The pip installing logic is not that sophisticated, it might overwrite existing config files. So please backup at least: `/etc/linotp2/linotp.ini` and `/etc/linotp2/encKey`!

If you have installed LinOTP using pip as described in *LinOTP Server Installation – the tar.gz, virtualenv and pip way* (page 109), you first can upgrade the main server components via the internet to the latest version:

```
pip install -upgrade linotp
```

Then upgrade the enterprise edition components. You need to download the newer version the customers portal:

```
pip install --upgrade /path/to/packages/LinOTP-EE-2.5.0.tar.gz
pip install --upgrade /path/to/packages/LinOtpUserIdResolverEE-2.5.0-2.tar.gz
```

10.3.3 Changelog

LinOTP Server

- Added OCRA token and QR-TAN functionality.
- Make TOTP token honor DefaultOTPLength configuration.
- Fixed bug, where a previous OTP value could be used again.
- Added support for DB2 Token database.
- Added framework of security modules to support HSMs to store the encryption keys.
- Added TOTP Google authenticator to self service .
- Improved SQLuserIdResolver (Performance).
- Improved LDAPResolver (entryUUID or ObjectGUID).
- Added passthru policy to authenticate users without token.
- Added client IPs to policies.
- Selfservice: added reset of failcounter.

11 | Migrating from LinOTP 1.3 or LinOTP 1.0

LSE provides a script to migrate token definitions with its user assignments from LinOTP 1.0 to LinOTP 2. On request this script can be easily enhanced to migrate tokens from LinOTP 1.3 to LinOTP. If you need assistance with migrating larger token numbers, drop us a note.

12 | Security advisories

12.1 Locked Users

When a user is locked in the userstore this information is not passed to LinOTP, as each userstore uses other means to lock a user. I.e. if you lock a user in Active Directory LinOTP will still authenticate this user successfully, when he provides the correct OTP value. If you also need to lock the user in LinOTP, you may lock all tokens of this user to disable his access. See the LSE customer portal for a small howto configure locked users [1].

12.2 UserIdResolver

The UserIdResolvers are used by the LinOTP Server to find the user object for a given loginname. The userstores are configured on the LinOTP Server. You can change this configuration with the LinOTP Management Client. Technically the client communicates with the LinOTP Server via HTTP and the URL path `/system/getConfig` and `/system/setConfig`. Although the data is stored encrypted in the database, in the current version the passwords for the LDAP Bind and the SQL user are transferred in plain text between the LinOTP Management Client and the LinOTP server. So assure, that

- you are using HTTPS,
- you are restricting access to the /system/ interface via the apache config
- you use an LDAP Bind that has only read access!

[1] https://www.lsexperts.de/kundenportal.html

13 | Troubleshooting

13.1 Connecting to userstores

If you have problems with the initial connection to your userstore, please check the log file `/var/log/linotp/linotp.log` for some occurrences like *module not found*. This may indicate, that you are missing some Python modules.

13.2 Distribution not Found: LinOTP2

The error message:

```
pkg_resources.DistributionNotFound: LinOTP
```

will appear, when the LinOTP Python modules are not installed into your system correctly. There might be a problem with your Python search paths. See section *Python search path* (page 117).

13.3 SQLAlchemy error

During installing or upgrading a LinOTP debian package you might get the error:

```
sqlalchemy.exc.OperationalError: (OperationalError) (1045, "Access denied for user 'linotp2'@'localhost' (using
```

This indicates that you either entered a wrong password for the linotp mysql user or that you changed the password manually. You can solve this problem by reinstalling:

```
dpkg -purge linotp
dpkg -i linotp-....
```

the LinOTP package.

13.4 Access denied for user root@localhost

During installing or upgrading a LinOTP debian package you might get the error:

```
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
```

During the first installation process you entered an SQL user, that did not exist yet. But you either entered a wrong root password or the installer process does not remember the root password correctly (in fact the installer process does not remember the root password, so it will use no password).

You can resolve this issue by:

```
dpkg -purge linotp
dpkg -i linotp-....
```

III LinOTP User Guide

1 | Introduction

This part shall present some usual workflows to the user.

2 | Workflows

2.1 Using mOTP token

LinOTP provides a self service interface that can be used by the user to register a new mOTP 31 token completely on his own. mOTP is a one time password algorithm. For this algorithm many different applications to run on mobile phones, smart phones and iPhone and iPad are available. Your Administration or IT department should have provided you the download link from where to install the mOTP application to your smartphone. In this workflow the MobileOTP.jar Java Midlet from http://motp.sourceforge.net is used.

2.1.1 Initializing the mOTP token

After installing the midlet to your phone, you need to initialize the application. Start the MobileOTP application.



Figure 2.1: *The icon to start the application on your phone.*

The OTP token can be initialized by entering the PIN “0000”. This can be repeated at any time afterwards.

Now you need to put in 25 random numbers, that are used to create the init secret.

Now the init secret is displayed. You should not write this down and not show it to any other, since this is the very secret that is used to calculate the OTP values. This secret is only displayed once. As soon as you enter the PIN, the secret can not be displayed anymore.

2.1.2 Registering the mOTP Token

You now need to open the LinOTP selfservice portal. Open your web browser and go to the address that was given to you by your IT department. It should be something like: https://linotp.yourdomain.com/ Then you need to login to the selfservice portal.

Here you should login using the credentials. This will be probably your domain credentials. For more details consult your IT department.

When successfully logged in, you are presented this screen:

At the start no tokens will be displayed on the left hand side. On the right hand side, you need to enter the Init-Secret, that is displayed on your phone. Also enter an mOTP PIN, that you will enter into the



Figure 2.2: *By entering the PIN '0000' the token can be initialized any time.*

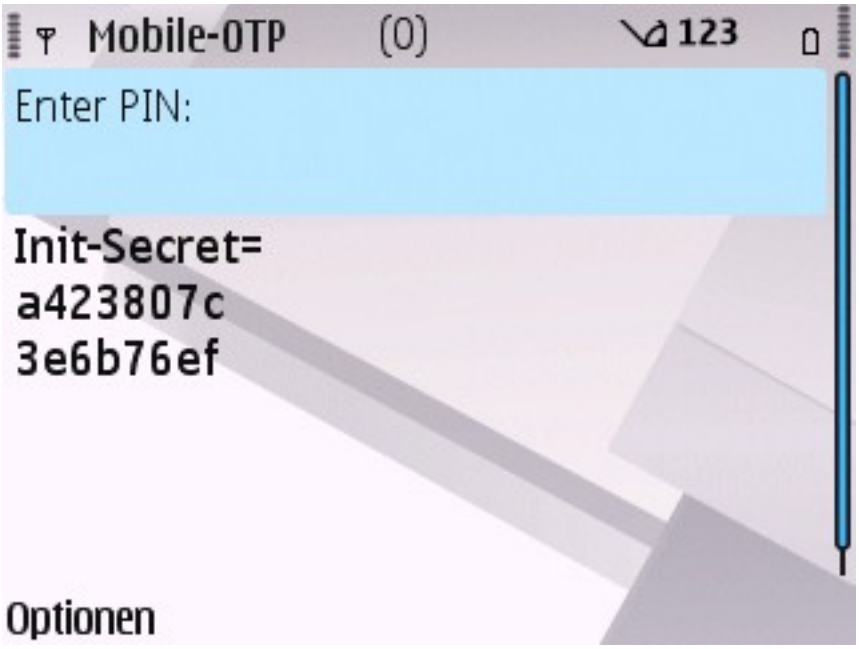


Figure 2.3: *The init-secret is only displayed once.*



Figure 2.4: *LinOTP selfservice portal login screen.*

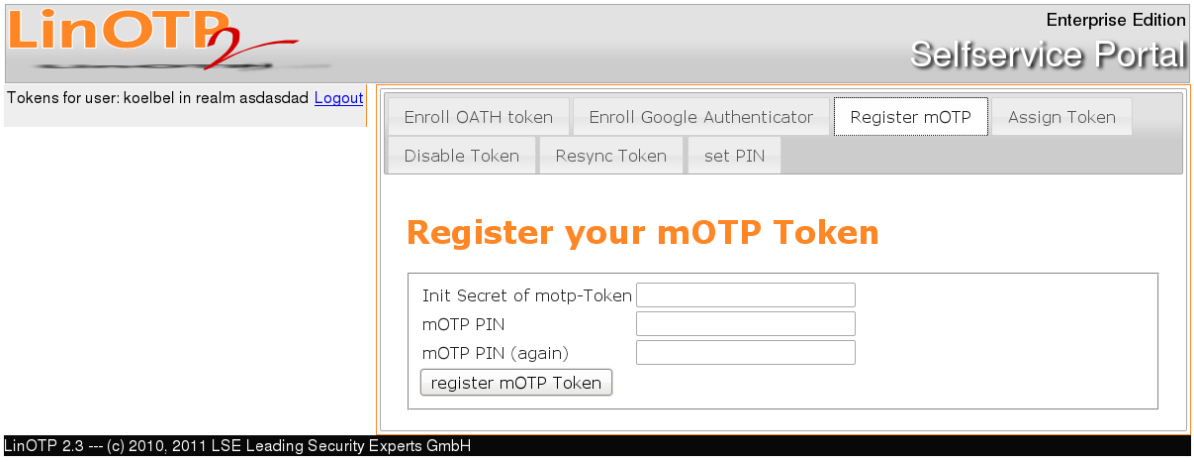


Figure 2.5: *LinOTP selfservice portal registering screen.*

MobileOTP application on your phone, each time you want to generate an OTP value. This mOTP PIN needs to be a 4 digit number.

When you press the button “register Token” your token data gets registered in the backend and assigned to your user. You will now see a token identifier on the left hand side.



Figure 2.6: *A token is assigned to the user.*

You may now set an additional OTP PIN.

This OTP PIN a fixed password, that is entered in front of the OTP value, each time you will authenticate. The OTP PIN can be an alpha numerical value. For this click on “set OTP PIN” and click on the token identifier on the left hand side.

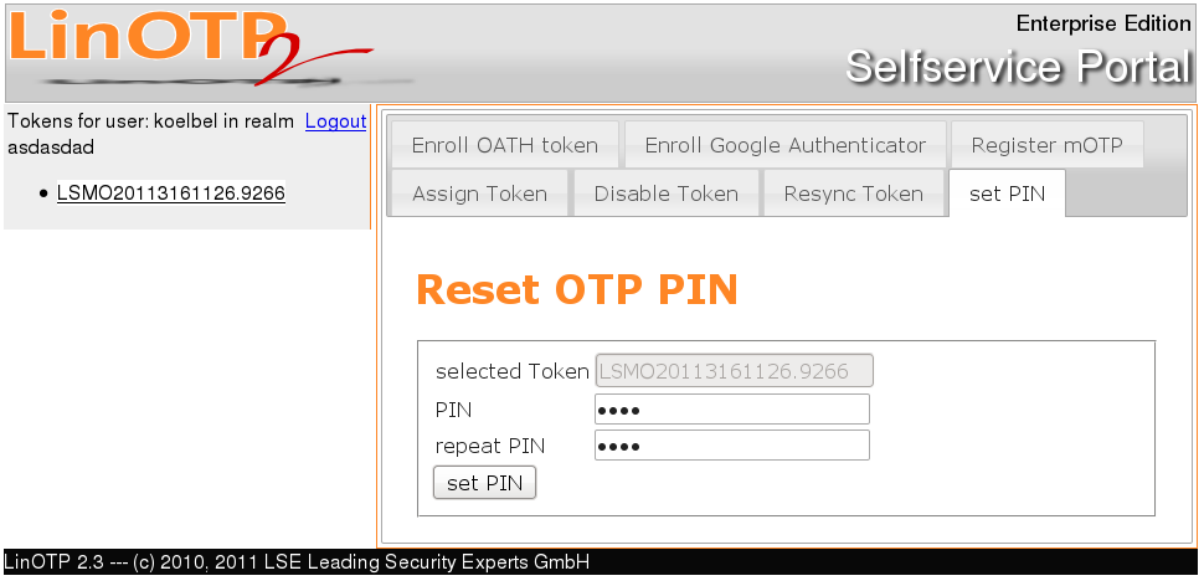


Figure 2.7: *Setting the OTP PIN for a token.*

Press the Button “set PIN” and log out.

2.1.3 Authenticating using mOTP Token

Probably you will use the mOTP token to authenticate to a web site, a VPN connection or to a terminalserver.

When doing so, you need to:

1. Enter your username into the login dialog username field
2. Enter your OTP PIN (the alpha numerical value) into the login dialog password field
3. Enter your mOTP PIN (the 4 digit number) into your MobileOTP application on your phone.
4. Your phone will display a one time password.
5. Now enter this one time password (0caa10) right behind the OTP PIN in the password field in the login dialog.
6. Press a button like “login”.



Figure 2.8: *Generated One Time Password.*

2.2 Using eToken Pass

Your IT department might have handed you an eToken PASS or Safeword Alpine Token.

Probably your IT department already assigned this token to your user account. Then you will only need to turn to the self service portal, if

- you lost your token,
- forgot your PIN or want to change your PIN for some reason
- or need to resynchronize the token.

2.2.1 Disable lost token

If you lost your token or left it somewhere so that someone else might probably use your token, you should go to the self service portal to disable your token. Please note, that only an administrator can enable the token again!





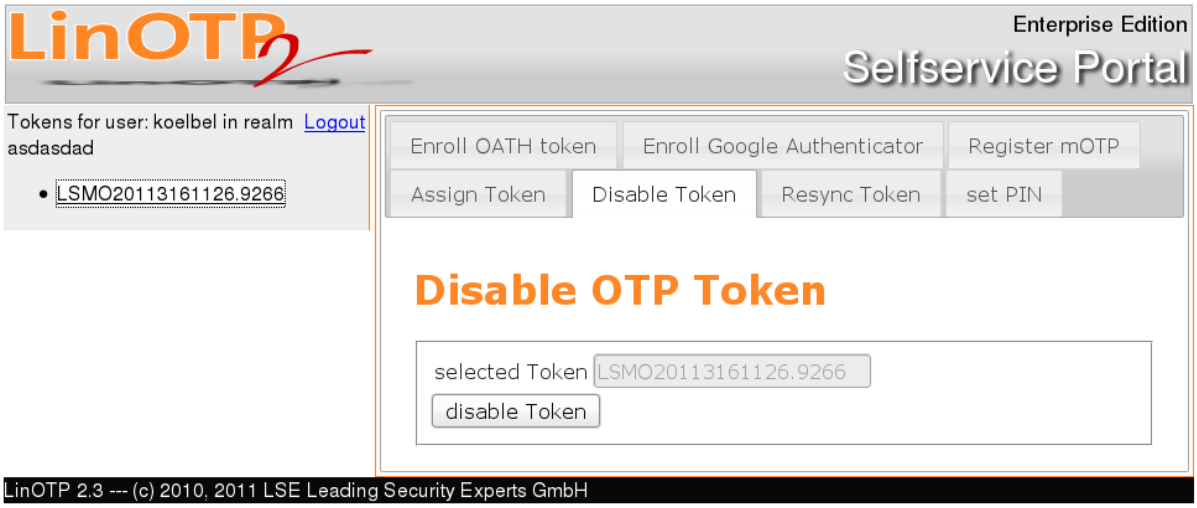


Figure 2.9: *Disabling a lost token.*

Choose “Disable Token” and select the token from the left side. Press the button “disable Token”. Logging in with this token will not be possible until it gets enabled by an administrator.

2.2.2 Change OTP PIN

If you forgot your OTP PIN or if you think, that someone spied on you and knows your OTP PIN, you can go to the self service portal to reset your OTP PIN.



Figure 2.10: *Reset OTP PIN.*

Select “set PIN” and choose the token of which you want to reset the OTP PIN from the left side. The serial number of the token will be displayed in the field “selected Token”. Enter a new OTP PIN two times and press the button “set PIN”.

2.2.3 Resynchronize token

As these tokens are event based tokens, you might get out of sync, if the button on a token is pressed to often without having authenticated successfully. In this case you can go to the self service portal to resynchronize your token.

Choose “resync Token” and select the token from the left side. Now you need to generate two successive OTP values with your token. Enter the first 6 digit OTP value in the field “OTP 1” and the second 6 digit OTP value in the field “OTP 2” and press the button “resync Token”.

2.2.4 Assign new token

If the IT department gave you a token without assigning it to you, you might go to the self service portal and assign it to your user.

If the token was not already assigned to you by the administrator, you need to flip the eToken PASS or Safeword Alpine token. On the backside you will find the serial number printed on a label. Choose “Assign Token” and enter this serial number in the field. After hitting the button “assign Token” this token will be assigned to you and appear on the left side in your token list.

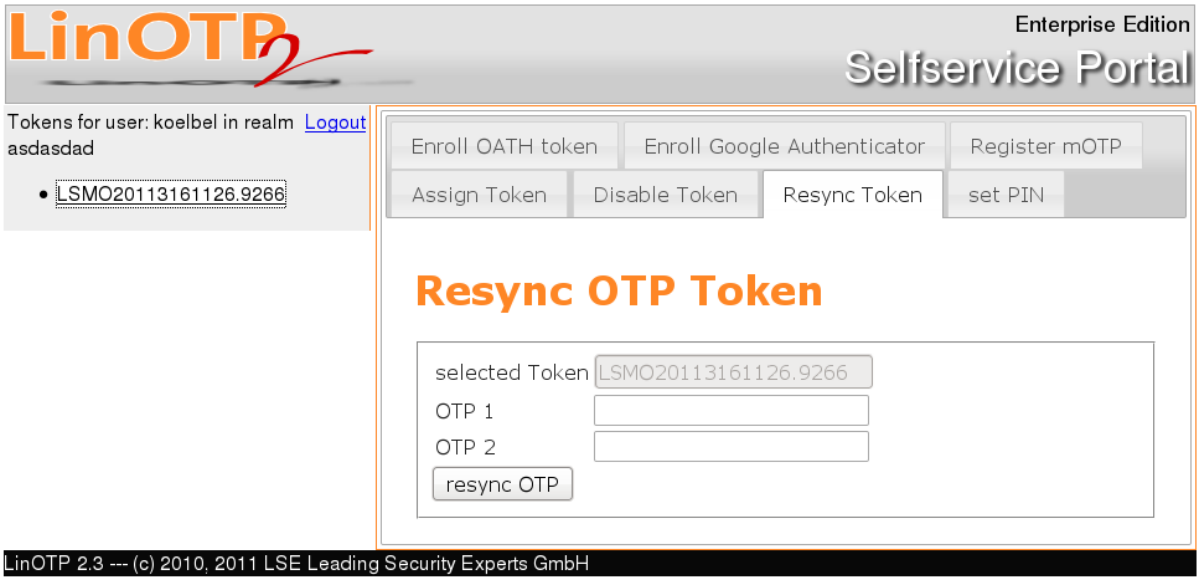


Figure 2.11: *Resynchronizing an eToken PASS.*

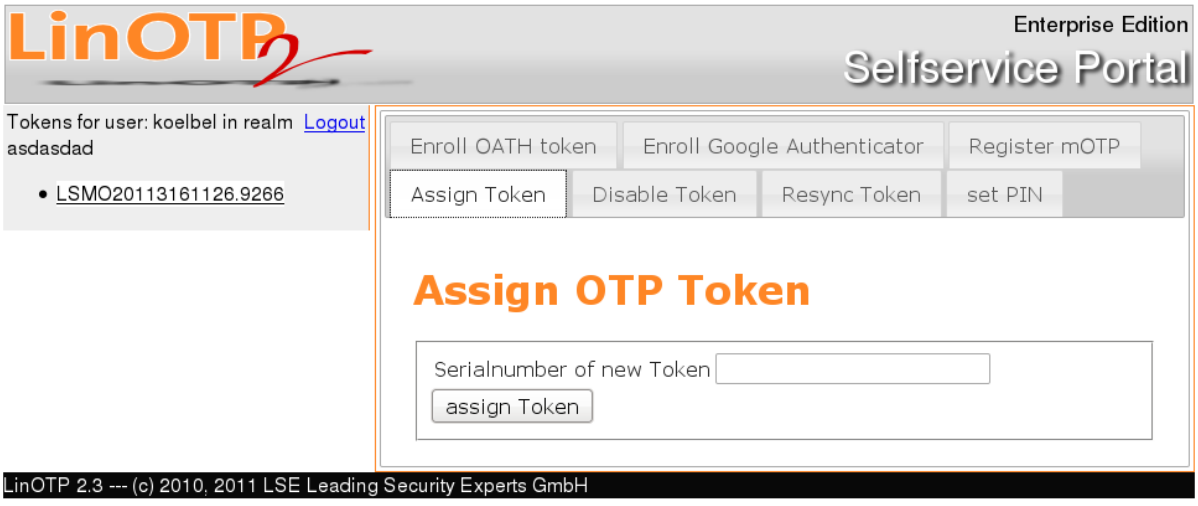


Figure 2.12: *Assign a new token.*

2.3 Enrolling OATH Token or Google Authenticator

LinOTP also supports the Google Authenticator, that is available for Android phones and iPhones and the “OATH Token” for iPhones.

These tokens can be easily enrolled using the two dimensional QR code. Install the Google Authenticator or OATH Token via app store. In the selfservice portal either choose “Enroll OATH token” or “Enroll Google Authenticator”, click on enroll and use the camera of your phone to scan the QR code picture.

IV LinOTP Appliance Manual

This part describes the interface and usage of the LSE LinOTP Appliance.

Note: Please note, that if using the Appliance some of the detailed configurations like some in the linotp.ini file are not possible. You may login to a restricted shell on the appliance using the root password. You can in fact raise your access rights to a full root shell, but then you will enter an unsupported area on the appliance, since we can not guarantee that the appliance will work as designed afterwards.

---

1 | Getting started

The appliance is managed using a Web UI, that can be reached via

https://10.76.83.69:8443

On the first run you need to configure your client machine to 10.76.83.70/30 and you will be taken to a configuration wizard.

The appliance management Web UI is used

- to configure the network settings of the machine,
- set passwords of the root user and LinOTP administrators
- perform backup and restore
- and update the machine.

You will use the Web UI to define a new appliance configuration. After having defined the configuration it needs to be saved and activated. You can revert to a previous configuration set at any time. The appliance keeps track of your configuration sets.

Let's take a look at the main tabs and thus at the main configuration topics.

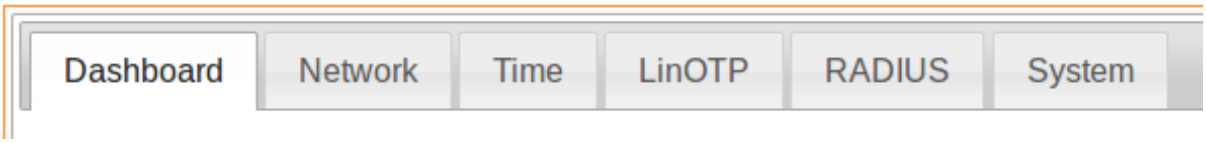


Figure 1.1: *Appliance main tabs*

`Dashboard` will give you an overview about the system status. Some graphics show on a timeline the memory consumption, the disk usage and other important information.

`Network` is used to configure all network settings including additional routes.

`Time` lets you set the time, the timezone and optional NTP server.

`LinOTP` is used to manage LinOTP token administrators and to set the debug level of the LinOTP application.

`RADIUS` is used to define RADIUS clients and other RADIUS settings.

`System` lets you save your configuration, manage certificates and passwords, backup and restore and all other basic tasks.

2 | Configuring network settings

The most important thing of course is to configure the network settings. The Appliance comes with a predefined fixed IP address. When connecting with a cross-over cable you can configure the IP settings, so that the appliance will fit into your network.

To perform the basic IP configuration got to the tab `Network` → `Interfaces`.

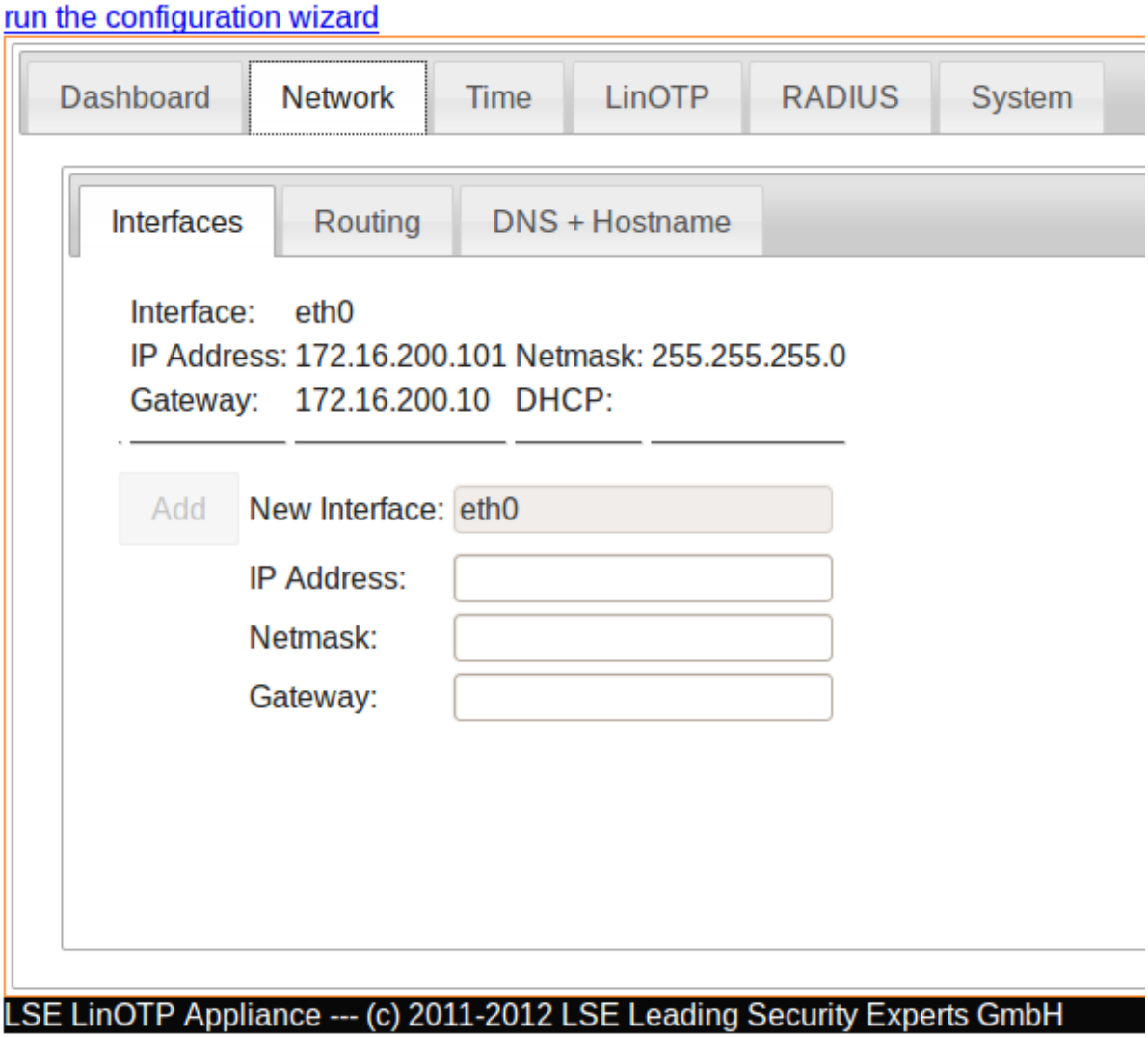


Figure 2.1: *Appliance Network Configuration*

Here you can setup the IP address, the netmask and the default gateway. Additionally you should go to `Network` → `DNS + Hostname` to setup the hostname and the DNS server.

If you need to setup additional routes go to `Network` → `Routing`.

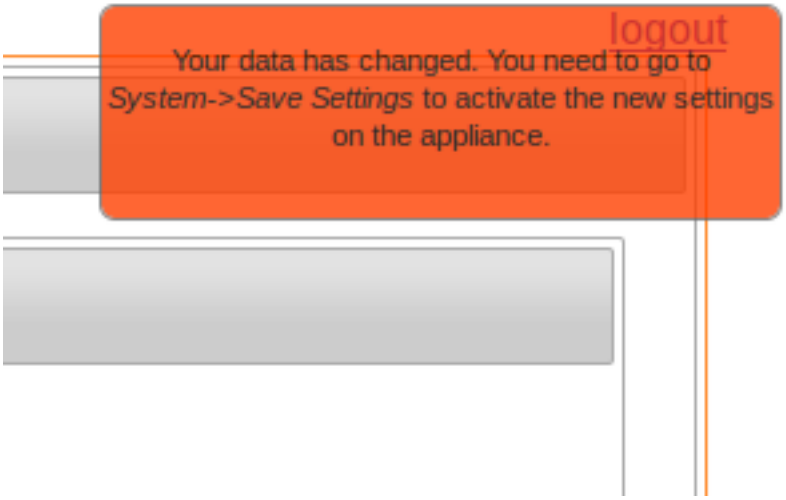


Figure 2.2: *A popup tells you that your configuration has changed and needs to be saved.*

Note: Each time you changed the current configuration set, you will get a note in the upper corner telling you, that the configuration was changed and that you should save and activate it. You can perform many configuration changes and only save them, when you think you are done.

---

3 | Managing LinOTP token administrators

On the tab `LinOTP` → `Administrators` new LinOTP token administrators can be added and the passwords of existing LinOTP token administrators can be changed.

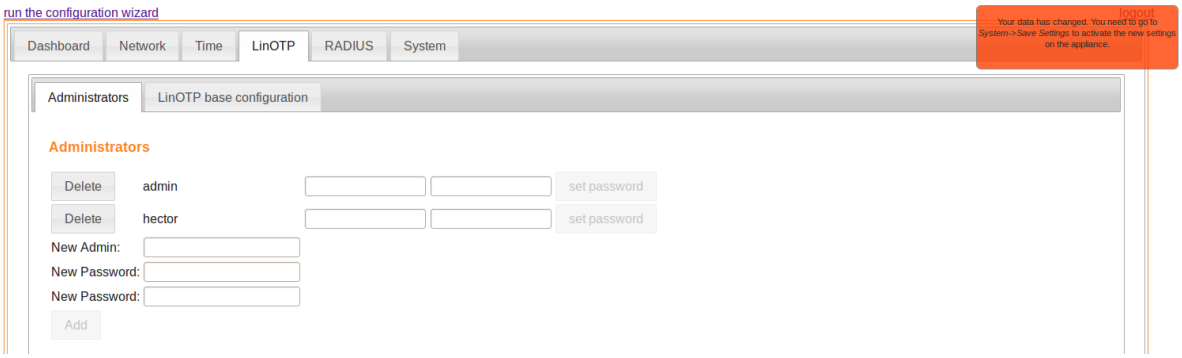


Figure 3.1: *Managing LinOTP token administrators*

Note: Only the Appliance administrators root and appadmin can change the passwords of the LinOTP token administrators.

Note: After adding, deleting or changing the LinOTP token administrators you need to activate the configuration as indicated by the orange box in the upper corner.

---

4 | LinOTP debug logging

In some cases the support might need some very detailed debug information. Then you can switch the log level of LinOTP to “debug”. This can be done on the tab `LinOTP` → `LinOTP base configuration`. After that you can download the support file, which contains the necessary log files on the tab `System` → `Support file`.

5 Configuring the RADIUS access to the LinOTP appliance

A RADIUS client is a computer that talks to the LinOTP server for authentication using the RADIUS protocol. This can be a SSL VPN, a firewall or the LSE RadiusGINA or Credential Provider.

You need to configure which RADIUS clients are allowed to contact the LinOTP Server via the RADIUS protocol. This can be done on the tab `RADIUS` → `Clients`. The client definition can contain a complete subnet. E.g. you can configure that the whole subnet 192.168.0.0/24 may access the LinOTP RADIUS server using a common secret. You can define many different RADIUS client definitions. Each should have a unique identifying name.

E.g if you want to have only one client access the LinOTP RADIUS server, you could specify 192.168.0.1/32, so that only the host 192.168.0.1 may access the RADIUS server.

Each RADIUS client definition may be given an optional `shortname`. This is an identifier you can use to refer to in the detailed RADIUS definitions.

5.1 Detailed settings

Detailed RADIUS settings can be defined on the tab `RADIUS` → `Settings`.

5.1.1 Realm mapping

Using the Realm mapping you can map a realm to RADIUS clients, i.e. you can define, that only a specified realm should be allowed to login to certain RADIUS clients.

This is a two-step process.

1. You need to map a realm to a RADIUS LinOTP instance. This will define a RADIUS internal instance, that can be used for further RADIUS policies.
2. In the second step you need to map the RADIUS client as defined by the shortname to this internal RADIUS instance.

5.1.2 Windows domain stripping

Some RADIUS clients will send the Windows domain with the RADIUS username. LinOTP usually only handles the username without FQDN. Some RADIUS clients may send

1. username@windows-domain (username@windows-domain)
2. NT-DOMAINusername
3. NT-DOMAIN/username

Note: LinOTP interprets the @-sign as a delimiter between username and LinOTP realmname. So if your Windows domain matches your LinOTP realm names, this can be an intended behavior.

6 | Working with configuration sets

The LinOTP server saves each configuration set before activating the configuration. To activate a configuration you need to go to the tab `System` → `Save Settings`. This tab will show you, which parts of the configuration were changed. You may give an optional tag for this revision of the configuration set.

The button `Save and Activate` will save the configuration and activate it.

You may restore any previous configuration set from the tab `System` → `Restore Settings`. Choose a previous revision. The revision will only get loaded into the management interface. Then you need to adapt it and `Save and Activate` this loaded revision, which in turn will create a new current revision, thus keeping track of your configuration without any gap.

7 | Root user and appadmin user

The users `root` and `appadmin` are system users known in the `/etc/passwd` file of the underlaying Linux operating system. These users can be used to login the Appliance Management UI.

While the user `root` can be used to login to the machine via console or SSH, the user `appadmin` is only allowed to login to the Management UI. Therefore the user `root` can change the password of the user `appadmin` but the user appadmin can not change the password of the user root. Both the users root and appadmin are allowed to change the passwords of all the LinOTP token administrators.

8 Change the server SSL certificate

The appliance comes with a self-signed SSL certificate for the HTTPS connection. On the tab `System → Password and Certificate` you can change the default certificate. Here you have got two options.

8.1 New self-signed certificate

You can create a new self-signed certificate if you do not trust the old certificate anymore.

8.2 Create Certificate Request – Externally signed Certificate

This is a two step process:

1. Create the certificate signing request. The hostname and domain name you entered in the network settings is used to make up the FQDN for the certificate signing request. Push the button `generate certificate signing request` and the copy the CSR to a file or directly to the CA where you want the request to be signed.
2. When you have got the certificate back from the CA paste the PEM encoded certificate and push the button `upload signed certificate`.

The private key does not leave the appliance. The uploaded certificate is checked to fit the private key.

9 | Advanced settings

On the tab `System` → `Advanced Settings` you can turn the SSH access to the appliance off.
You can reboot and shutdown the appliance and you can create a new encryption key.

Note: Creating a new encryption key will make your previous token data unreadable.

---

10 | Redundant setup

Two appliances may be setup in a redundant scenario based on the MySQL master-master-replication.

On an existing `Appliance A` the setup is started and pushed to an fresh `Appliance B`. After a successful setup the data is synchronized from `A` to `B` and vice versa. The communication between the appliances will be encrypted using IPSec.

Note: As the communication between the two redundant partners is done via IPSec, please assure that IPSec between the two partners is possible!

The Appliance Web UI shows the status of the configured redundancy.

10.1 Setup redundancy

Note: During the setup of the redundancy the webserver needs to be restarted. So - to make the setup more robust - the setup is started from the command line.

Warning: All token data on `Appliance B` will be lost!

Run the following steps:

1. Login to your `Appliance A` and issue the following command:

```
appliance_configure.py -c setup_redundancy -p <IP of Appliance B>

Welcome to the redundancy wizard.
Please make sure that timer_entropyd is running.
Do you want to setup IPSec-based encryption between the machines? [y/n] y
You will have to verify the partner hostkey fingerprint and give the partner's root password in order to s
Since the SSH command only waits for input for a certain time, please make sure that:
 * the partner hostkey fingerprint
 * the partner root password
are close at hand.
You can find the partner hostkey fingerprint at http://10.76.126.196:8443/ at System-Advanced-Redundancy.
Please hit RETURN to continue:
```

   The SSH fingerprints of each appliance can also be viewed in the Appliance Web UI. Please assure that you have the root password and the fingerprint of `Appliance B` available.

2. Now you are asked to confirm, that the fingerprint of `Appliance B` is correct:

```
The authenticity of host '10.76.126.196 (10.76.126.196)' can't be established.
RSA key fingerprint is 4c:bc:02:9f:e8:27:01:bc:64:c7:6e:0e:bc:cb:5e:1a.
Are you sure you want to continue connecting (yes/no)?
```

Note: If the fingerprints do not match, this is not your `Appliance B` you are talking to! Please contact your reseller, you might be the victim of a man-in-the-middle attack!

If the fingerprints match, you can enter the root password of `Appliance B`.

3. Now all necessary data are transferred from `Appliance A` to `Appliance B` via SSH. Somme messages are displayed during the transfer. If any error occurs you can check the logfile `/var/log/lseappliance/appliance.log`.
4. Finally you will get the message:

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.76.126.196' (RSA) to the list of known hosts.
root@10.76.126.196's password:
Success!
Terminating the SSH tunnel ... (this is expected!)
```

At this point the redundancy is setup successfully and the communication for the replication is done encrypted via IPSec.

Congratulations, you are done!

10.2 Known Errors

There is a timeout when entering the root password of `Appliance B`. Then you might get a message like this:

```
A SSH error occurred:
---- BEGIN ----
Warning: Permanently added '10.76.126.196' (RSA) to the list of known hosts.
Connection closed by 10.76.126.196
---- END ----
It's possible that this happened due to the SSH server timeout.
Since the redundancy setup process hasn't been started yet, it's safe
to re-run `appliance_configure.py -c setup_redundancy -p 10.76.126.196`.
```

You can just rerun the setup.

10.3 Reverting Redundancy

If you want to quit using the redundancy, you need to execute this command on *both* appliances:

```
appliance_configure.py -c reset_redundancy
```

Now both appliances will work as a single appliance without knowing the other partner anymore.

11 | The support file

The support file can be downloaded from the tab `System` → `Support`. This file contains all log files from the appliance and the basic configuration files. Under certain circumstances a support engineer will ask you to create and download such a support file.

12 | Updates

The appliance retrieves its updates from a download server from LSE Leading Security Experts GmbH. An update key is used to authenticate the appliance on the update request.

A hardware appliance uses an internal update key you do not need to worry about. If you are running a virtual appliance, the updates are based on the serial number, you entered during installation.

On the tab `System` → `Update` you can specify when the system should look for updates automatically and if it should reboot when the kernel was updated. As kernel updates can happen every time, you can also configure to boot the machine automatically on certain days of the week.

13 | Backup and restore

You can perform backup and restore using the tabs `System` → `Backup` and `System` → `Restore`.

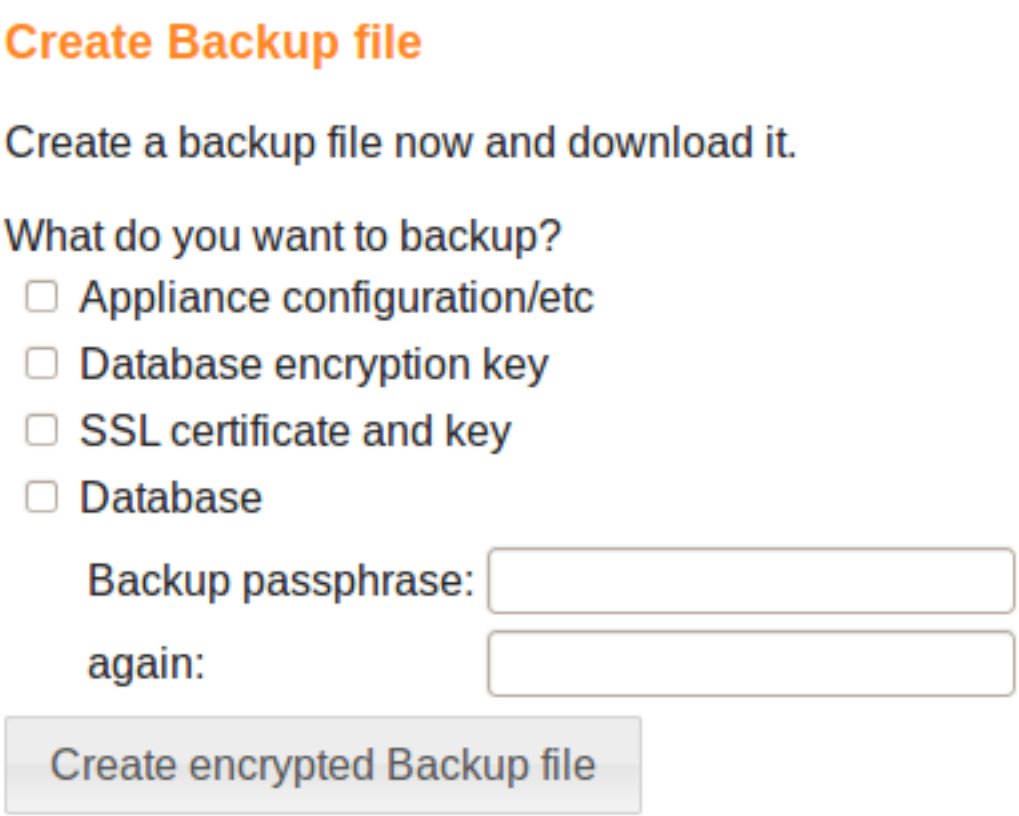


Figure 13.1: *Create a backup file.*

In any case you can choose what you want to backup and what you want to restore. Each backup file is encrypted with AES-256 with an AES key derived from a passphrase you choose. A backup file may contain

- the appliance configuration files contained in /etc. This also contains the configuration sets,
- the database encryption key,
- the SSL certificate,
- the database.

At the moment only things that can be restored are backed up. There is no way to backup e.g. log files.

You may also perform scheduled automatic backups. Those backups are automatically stored to a Windows share on your file server.

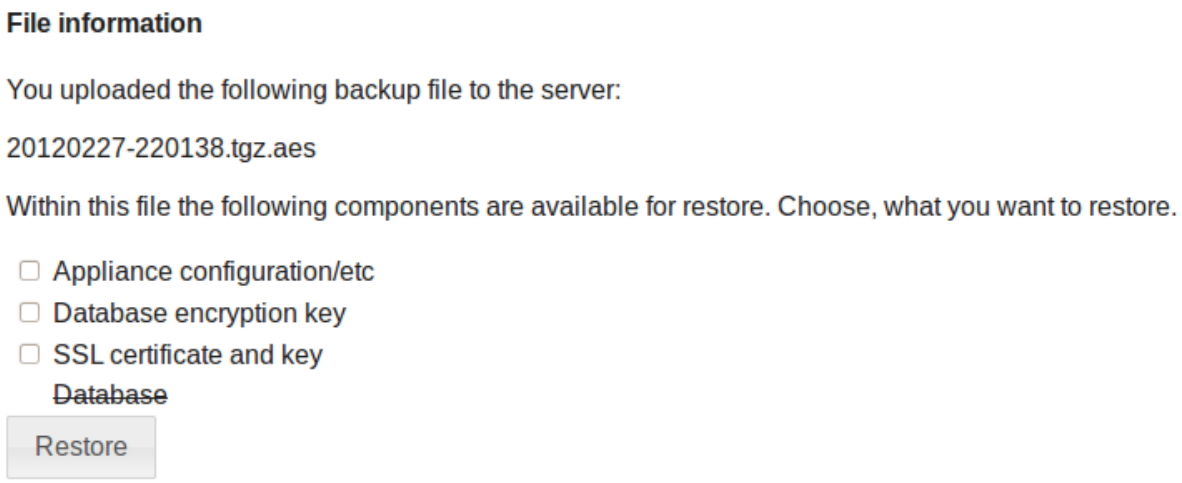


Figure 13.2: *Restore a backup file without the database.*

In case of restore a backup file is uploaded, the encryption passphrase is passed and the appliance unpacks the backup file. It then determines what is contained in the backup file. In the example in figure *Restore a backup file without the database.* (page 222) the backup file contains everything except the database. Now you can choose which of the contained components you want to restore.

14 | Network integration

If you want to place the LSE LinOTP Appliance inside a DMZ you need to take care about the following ports for communication.

Purpose	Source	Destination	Dest-Port
Authentication	SSL-VPN or other Application	Appliance	1812 UDP
Management	Client	Appliance	443 TCP, 8443 TCP
UserIdResolver	Appliance	SQL Database	depends on database
UserIdResolver	Appliance	LDAP, Active Directory	389 TCP, 636 TCP
Updating Appliance	Appliance	linotp-appliance.lsexperts.de	80 TCP
NTP	Appliance	Time server, e.g. domain controller	123 UDP
DNS	Appliance	DNS server, e.g. domain controller	53 UDP

15 | Emergency recovery

The Appliance provides an Emergency Recovery. You need to connect via the serial console to the appliance to can start the recovery process. During this process the appliance connects to the LSE server and installs completely from the scratch.

This method can be used if the appliance is completely misconfigured and will not boot anymore.

Note: LSE provides an additional documentation for the Emergency Recovery process. Please contact your Reseller or Distributor to help you with such a scenario.

V LinOTP Module Development Guide

This part describes the interfaces you need to know, if you plan to develop further modules for LinOTP.

1 Authentication interfaces

There are different authentication controllers.

`ValidateController` This controller is used to authenticate simple credentials of event or timebased tokens. I.e. the controller has functions like `check`, `check_s`, `simplecheck`... to basically take a username and password to authenticate this user.

`OcraController` This controller is used for challenge response tokens following the OCRA standard. The controller also provides a function `request` to get the challenge but also a function `check_t` to validate the response to the challenge.

The methods are called as a HTTP GET request and the described parameters are added as HTTP parameters.

The response of the HTTP Request will usually be a JSON object.

Note: To every authentication controller you can add the parameter `httperror` followed by an HTTP error code. If LinOTP would return HTTP 200/OK with `status: false` in the JSON response indicating an internal error then LinOTP will instead return a e.g. HTTP 500 error code.

1.1 Validate Controller

class `linotp.controllers.validate.ValidateController`(**args*, ***kw*)

The linotp.controllers are the implementation of the web-API to talk to the LinOTP server. The ValidateController is used to validate the username with its given OTP value. An Authentication module like pam_linotp2 or rlm_linotp2 uses this ValidateController. The functions of the ValidateController are invoked like this

https://server/validate/<functionname>

The functions are described below in more detail.

`check()`

This function is used to validate the username and the otp value/password.

method: validate/check

arguments:

- user: The username or loginname
- pass: The password that consist of a possible fixed password component and the OTP value
- realm (optional): An optional realm to match the user to a useridresolver
- challenge (optional): optional challenge + otp verification for ocra tokens

returns: JSON response:

```
{
    "version": "LinOTP 2.4",
    "jsonrpc": "2.0",
    "result": {
        "status": true,
        "value": false
    },
    "id": 0
}
```

If `status` is `true` the request was handled successfully.

If `value` is `true` the user was authenticated successfully.

`simplecheck()`
This function is used to validate the username and the otp value/password.

method: validate/simplecheck

arguments:

- user: username / loginname
- **pass: the password that consists of a possible fixes password component** and the OTP value
- realm: additional realm to match the user to a useridresolver

returns: Simple ascii response:

:-) in case of success

:-(in case of failed authentication

:-/ in case of any error

`check_s()`
This function is used to validate the serial and the otp value/password.

method: validate/check_s

arguments:

- serial: the serial number of the token
- **pass: the password that consists of a possible fixes password component** and the OTP value

returns: JSON response

`smspin()`
This function is used in conjunction with an SMS token: the user authenticates with user and pin (pass) and will receive on his mobile an OTP as message

method: validate/smspin

arguments:

- user: username / loginname
- pass: the password that consists of a possible fixed password
- realm: additional realm to match the user to a useridresolver

returns: JSON response

`samlcheck()`
This function is used to validate the username and the otp value/password in a SAML environment. If `linotp.allowSamlAttributes = True` then the attributes of the authenticated users are also contained in the response.

method: validate/samlcheck

arguments:

- user: username / loginname
- pass: the password that consists of a possible fixes password component and the OTP value
- realm: optional realm to match the user to a useridresolver

returns: JSON response

1.1.1 Authentication workflow

The following diagram denotes the authentication workflow. The loginname is reverse resolved to the userid by the useridresolver. Using the userid all tokens assigned to this user can be found in the token database.

1.2 OCRA Controller

Most functions of the OCRA controller provide insight into the interna of the OCRA token, this is why the OCRA controller is also protected via authentication. Additionally the administrator or a system account needs to have a policy defined, to be able to access the OCRA functions. See *OCRA Policies* (page 59).

The only function, that can be accessed directly is the *ocra/check_t* function.

class `linotp.controllers.ocra.OcraController(*args, **kw)`

The OcraController implements challenges/response tokens according to RFC 6287

`request()`

method: orcra/request

description: request a challenge for a user or for a serial number (token).

arguments:

- **serial: (required - string)** Serial number of the token, for which a challenge should be generated (either serial or user is required)
- **user: (required - string)** The user for whose token a challenge should be generated If the user has more than one token, an error is returend. (either serial or user is required)
- **data: (required - String: URLendoced)** These are the display data, that can be used to generate the challenge

remark:

the app will report a wrong qrcode, if the policy {'authentication' : qrtanurl=https://localhost } is not defined !!

returns:

A JSON respone:

```
{
    "version": "LinOTP 2.4",
    "jsonrpc": "2.0",
    "result": {
        "status": true,
        "value": false,
```

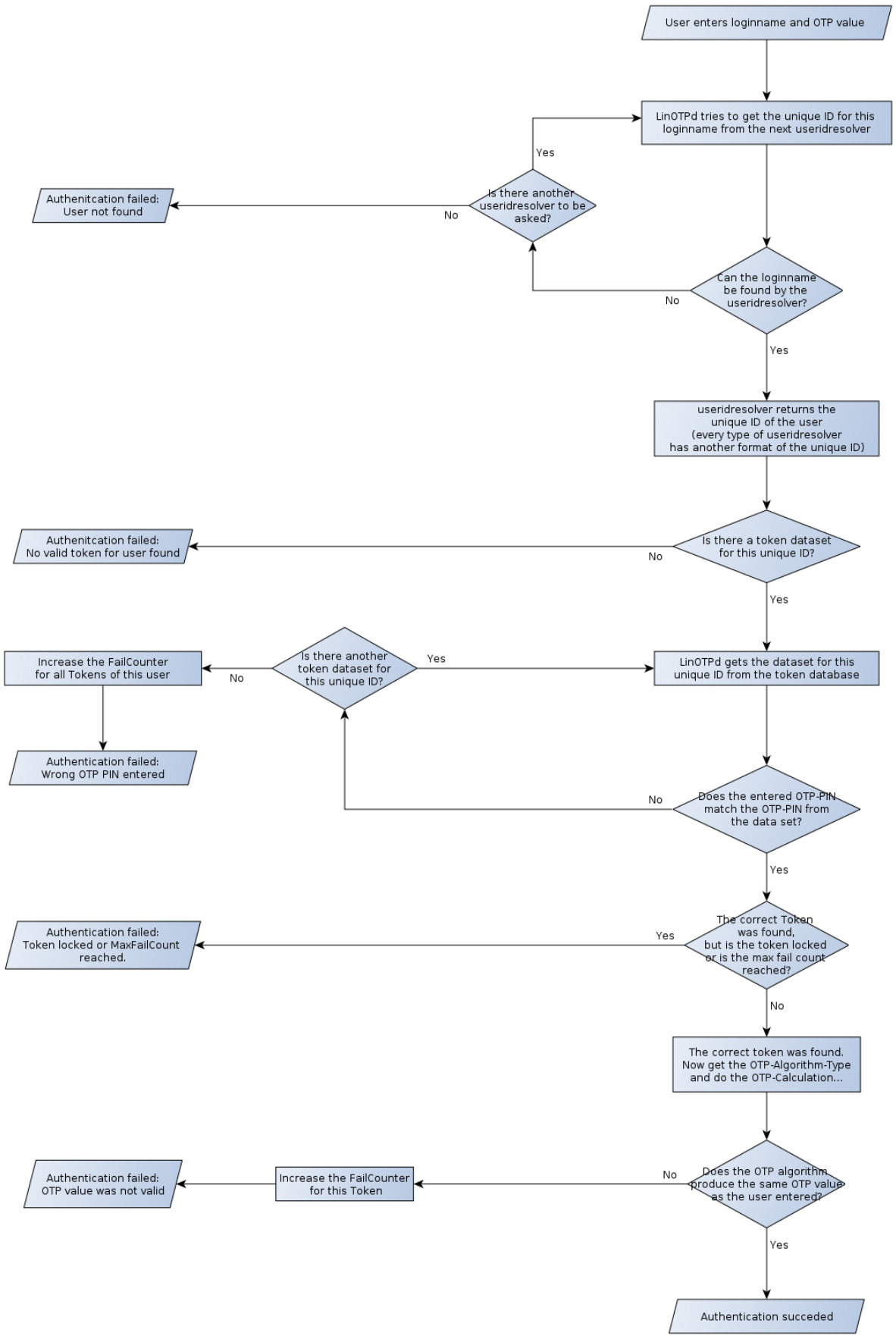


Figure 1.1: *The authentication workflow in LinOTP*

```
        },
        "detail": {
                "transactionid" : TRANSAKTIONSID,
                "data" : DATAOBJECT,
        }
    }
```

- **transactionid:** This is the transaction ID, that is used later for verifying the Return code /TAN.
- **data:** This is an object (URL) which can be used to generate a QR-Coide to be displayed to the QRTAN App

exception:

`check_t()`

method: orcra/check_t

description: verify the response of the ocra token

arguments:

- **transactionid: (required - string)** Dies ist eine Transaktions-ID, die bei der Challenge ausgegeben wurde.
- **pass: (required - string)** die response, die der OCRA Token auf Grund der Challenge berechnet hat

returns:

A JSON response:

```
{
 "version": "LinOTP 2.4",
 "jsonrpc": "2.0",
 "result": {
     "status": true,
     "value": {
         "failcount" : 3,
         "result": false
        }
    },
 "id": 0
}
```

exception:

`checkstatus()`

method: orcra/checkstatus

description: Methode zur assynchronen Ueberpruefungen eines Challenge Response Validation requests

arguments:

- **transactionid: (required one of - string - (hex))** Dies ist eine Transaktions-ID, die bei der Challenge ausgegeben wurde.
- **serial: (required one of - string)** die Serien Nummer des OCRA Token
- **user: (required one of - string)** die Benutzer eines Tokens

required is one of (user,serial,transactionid)

returns:

A JSON response:


```
{
 "version": "LinOTP 2.4",
 "jsonrpc": "2.0",
 "result": {
     "status": true,
     "value": [
             {
             "serial": SERIENNUMMER1,
             "transactionid": TRANSACTIONID1,
             "received_tan": true,
             "valid_tan": true,
             "failcount": 0
             },
             {
             "serial": SERIENNUMMER1,
             "transactionid": TRANSACTIONID2,
             "received_tan": false,
             "valid_tan": false,
             "failcount": 0
             },
             {
             "serial": SERIENNUMMER2,
             "transactionid": TRANSACTIONID3,
             "received_tan": true,
             "valid_tan": false,
             "failcount": 2
             },
         ]
     },
 "id": 0
 }
```


exception:

`getActivationCode()`

method: orcra/getActivationCode

description: returns an valid example activcation code

arguments: ./.

returns: JSON with “activationcode”: “JZXW4ZI=2A”

`calculateOtp()`

1.3 Example for authentication integration

Adding LinOTP authentication to your existing source code is quite simple.

You will need to use the API as described in section *Validate Controller* (page 231).

Both examples use cURL [1] to talk to the LinOTP server.

1.3.1 C integration

For usual binary programs written in C, you could take a look at the pam_linotp implementation (file pam/src/pam_linotp.c) which can be found here: http://linotp.org/download/LinOTP-2.x-CE.tar.gz.

[1] http://curl.haxx.se

This could look like this to create the URL, that needs to be sent:

```
char * createUrl(CURL *curl_handle,
    const char * validateurl,
    const char * realm, const char * resConf,
    const char * user, const char * password)
{
 char *   url    = NULL;
 int size    = 300;
 int nchars     = 0;

 // escape user and password
 char *escPassword;
 char *escUser;
 escPassword    = curl_easy_escape(curl_handle, password, 0);
 escUser     = curl_easy_escape(curl_handle, user, 0);
 char *escRealm = curl_easy_escape(curl_handle, realm, 0);
 char *escResConf     = curl_easy_escape(curl_handle, resConf, 0);

 if (escPassword == NULL || escUser == NULL)
 {
        log_error("ERROR: failed to escape user or password");
        goto cleanup;
 }

    url = (char*) malloc (size);
    if (url == NULL)
    {
        log_error("ERROR: could allocate size for url");
        goto cleanup;
    }

 // allocate the memory for url string
 memset(url,'\0',size);
 nchars = snprintf( url, size-1, "%s?user=%s&pass=%s&realm=%s&resConf=%s",
                validateurl,
          escUser, escPassword,
    escRealm, escResConf );
 if (nchars >= size-1)
 {
    // reallocate
    size = nchars +1;
    url = (char*) myrealloc (url, size);
    if (url == NULL)
    {
        log_error("ERROR: failed to alloc space for url + user and password");

            goto cleanup;
    }

    memset(url,'\0',size);
    snprintf(url, size-1, "%s?user=%s&pass=%s&realm=%s&resConf=%s", validateurl,
            escUser, escPassword,
            escRealm, escResConf );
 }

cleanup:
 return url;
}
```

The authentication itself could look like this:

```
int linotp_auth(const char *user, const char *password, const char *validateurl,
    const char *realm, const char *resConf,
```

```
    int nosslhostnameverify, int nosslcertverify, int debug)
{
 CURL *curl_handle       = NULL;
 int    returnValue         = PAM_AUTH_ERR;
 char *url               = NULL;
 CURLcode all_status        = 0;

 char errorBuffer[CURL_ERROR_SIZE];

 struct MemoryStruct chunk;
    chunk.memory      = NULL; /* we expect realloc(NULL, size) to work */
    chunk.size        = 0;    /* no data at this point */

    curl_global_init(CURL_GLOBAL_ALL);

    curl_handle = curl_easy_init();

    if (curl_handle == NULL)
    {
        log_error("ERROR: could not get curl_handle!");
       returnValue = PAM_AUTH_ERR;
       goto cleanup;
    }


    curl_easy_setopt(curl_handle, CURLOPT_ERRORBUFFER, errorBuffer);

    url = createUrl(curl_handle, validateurl, realm, resConf, user, password);
 if (url == NULL)
 {
        log_error("ERROR: could allocate size for url");
    goto cleanup;
 }

 if (debug)
 {
    log_debug("DEBUG: connecting to url:%s",url);
 }

 all_status = sendRequest(curl_handle, url,
       (void *)&chunk, nosslhostnameverify, nosslcertverify);

  /* set default return*/
 returnValue = PAM_AUTH_ERR;

 if (all_status != 0)
 {
    log_error("ERROR: Error talking to linotpd server at %s: %s", url, errorBuffer);
    returnValue = PAM_AUTH_ERR;
    goto cleanup;
 }
 if(chunk.memory == NULL)
 {
    log_error("ERROR: No response returned for %s: %s", url, errorBuffer);
    returnValue = PAM_AUTH_ERR;
    goto cleanup;
 }
 if (strcmp(chunk.memory, LINOTPD_REJECT) == 0)
 {
    log_warning("WARNING: user '%s' rejected", user);
    returnValue = PAM_AUTH_ERR;
    goto cleanup;
 }
```

```
 if (strcmp(chunk.memory, LINOTPD_FAIL ) == 0)
 {
    log_warning("WARNING: authentication for '%s' failed", user);
    returnValue = PAM_AUTH_ERR;
    goto cleanup;
 }
 if (strcmp( chunk.memory, LINOTPD_OK ) == 0)
 {
    log_info("INFO: user '%s' authenticated successfully\n", user);
    returnValue = PAM_SUCCESS;
    goto cleanup;
 }
 // default
 {
    log_error("ERROR: An error occurred for '%s' on '%s'\n", user, validateurl);
    returnValue = PAM_AUTH_ERR;
    goto cleanup;
 }


cleanup:

 if (chunk.memory!= NULL)
 {
    free(chunk.memory);
 }
 if (url != NULL)
 {
    free(url);
 }

    /* we're done with libcurl, so clean it up
       cleanup also takes care of escPassword and escUser
    */
 curl_global_cleanup();
 if (debug)
 {
    log_debug("linotp_auth exited normally.");
 }
 return (returnValue);
}
```

In this example we defined a helper function `sendRequest`:

```
int sendRequest(CURL *curl_handle, char * url,
    struct MemoryStruct * chunk,
    int nosslhostnameverify, int nosslcertverify)
{
 int all_status = 0;
 int status     = 0;

 all_status  = 0;
 status      = curl_easy_setopt(curl_handle, CURLOPT_URL, url);
 all_status += status;

 status      = curl_easy_setopt(curl_handle, CURLOPT_WRITEFUNCTION, WriteMemoryCallback);
 all_status += status;

 status      = curl_easy_setopt(curl_handle, CURLOPT_WRITEDATA, chunk);
 all_status += status;

 status      = curl_easy_setopt(curl_handle, CURLOPT_USERAGENT, "libcurl-agent/1.0");
 all_status += status;
```

```
 if ( nosslhostnameverify )
    status = curl_easy_setopt(curl_handle, CURLOPT_SSL_VERIFYHOST, 0L);
 else
    status = curl_easy_setopt(curl_handle, CURLOPT_SSL_VERIFYHOST, 2L);
 all_status += status;

 if ( nosslcertverify )
       status = curl_easy_setopt(curl_handle, CURLOPT_SSL_VERIFYPEER, 0L);
 else
       status = curl_easy_setopt(curl_handle, CURLOPT_SSL_VERIFYPEER, 1L);
 all_status += status;

 status      = curl_easy_perform(curl_handle);
 all_status += status;

 curl_easy_cleanup(curl_handle);

 return all_status;
}
```

1.3.2 PHP integration

For usual web applications or portals that might be written in PHP you could use code like this:

```
<?php
public function linotp_auth($user="", $pass="", $realm="") {
   $ret=false;
   try {
      $server = $this->server;
      $REQUEST="https://$server/validate/check?pass=$pass&user=$user";
      if (""!=$realm)
         $REQUEST="$REQUEST&realm=$realm";
      if (!function_exists("curl_init"))
         die("cURL extension is not installed!");
      $ch=curl_init($REQUEST);
      curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
      curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, $this->verify_peer);
      curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, $this->verify_host);
      $r=curl_exec($ch);
      curl_close($ch);

      $Object = json_decode($r);
      if (true == $Object->{'result'}->{'status'} )
         if (true == $Object->{'result'}->{'value'} )
            $ret=true;
   } catch (Exception $e) {
      print "Error in receiving response from LinOTP server: $e";
   }
   return $ret;
}
?>
```

2 | Administrative Interfaces

Note: To every management controller you can add the parameter `httperror` followed by an HTTP error code. If LinOTP would return HTTP 200/OK with `status: false` in the JSON response indicating an internal error then LinOTP will instead return a e.g. HTTP 500 error code.

2.1 Admin Interface

Managing tokens can be performed via an HTTPs Interface just like the authentication via /validate/check and /validate/simplecheck. The main API is located at the controller /admin. There you can find functions like this:

- /admin/init: Enroll token
- /admin/enable: Enable token
- /admin/disable: Disable token
- /admin/show: List tokens
- /admin/delete: Delete tokens

Other controllers are

- /manage: provides the Web UI.
- /system: provides the functions to configure the LinOTP server.
- /license: provides the functions to set and get license information.
- /audit: provides the functions for the audit trail.

2.1.1 Orphanced tokens

Orphaned tokens are tokens, that are still assigned to the user, but where the user object was deleted from the user database.

Using the method `/admin/show` you can also view such orphaned tokens by calling the method:

```
/admin/show?user='/:no%20user%20info:/'
```

You can also find tokens that have no users or tokens that are assigned to no realm:

```
/admin/show?user=''
```

```
/admin/show?realm=''
```

2.2 Admin Controller

class `linotp.controllers.admin.AdminController`(**args*, ***kw*)

The linotp.controllers are the implementation of the web-API to talk to the LinOTP server. The AdminController is used for administrative tasks like adding tokens to LinOTP, assigning tokens or revoking tokens. The functions of the AdminController are invoked like this

https://server/admin/<functionname>

The functions are described below in more detail.

`assign()`

method: admin/assign

description: assigns a token to a user, i.e. a binding between the token and the user is created.

arguments:

- serial - required - the serial number / identifier of the token
- user - required - login user name
- pin - optional - the pin of the user pass

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`check_serial()`

method admin/check_serial

description: This function checks, if a given serial will be unique. It returns True if the serial does not yet exist and new_serial as a new value for a serial, that does not exist, yet

arguments: serial - required- the serial to be checked

returns: a json result with a new suggestion for the serial

exception: if an error occurs an exception is serialized and returned

`copyTokenPin()`

method: admin/copyTokenPin

description: copies the token pin from one token to another

arguments:

- from - required - serial of token from
- to - required - serial of token to

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`copyTokenUser()`

method: admin/copyTokenUser

description: copies the token user from one token to another

arguments:

- from - required - serial of token from
- to - required - serial of token to

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`disable()`

method: admin/disable

description: disables a token given by serial or all tokens of a user

arguments:

- serial - optional
- user - optional

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`enable()`

method: admin/enable

description: enables a token or all tokens of a user

arguments:

- serial - optional
- user - optional

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`getSerialByOtp()`

method: admin/getSerialByOtp

description: searches for the token, that generates the given OTP value. The search can be restricted by several critterions

arguments:

- otp - required. Will search for the token, that produces this OTP value
- type - optional, will only search in tokens of type
- realm - optional, only search in this realm
- assigned - optional. 1: only search assigned tokens, 0: only search unassigned tokens

returns: a json result with the serial

exception: if an error occurs an exception is serialized and returned

`getsession()`

This generates a session key and sets it as a cookie set_cookie is defined in python-webob:

def set_cookie(self, key, value='', max_age=None, path='/', domain=None, secure=None, httponly=False, version=None, comment=None, expires=None, overwrite=False):

`init()`

method: admin/init

description: creates a new token.

arguments:

- otpkey - required - the hmac Key of the token
- genkey - required - =1, if key should be generated. We either need otpkey or genkey
- keysize - optional - either 20 or 32. Default is 20
- serial - required - the serial number / identifier of the token
- description - optional
- pin - optional - the pin of the user pass
- user - optional - login user name
- realm - optional - realm of the user
- type - optional - the type of the token
- tokenrealm - optional - the realm a token should be put into
- otplen - optional - length of the OTP value
- hashlib - optional - used hashlib sha1 oder sha256

ocra arguments: for generating OCRA Tokens type=ocra you can specify the following parameters:

- ocrasuite - optional - if you do not want to use the default ocra suite OCRA-1:HOTP-SHA256-8:QA64
- sharedsecret - optional - if you are in Step0 of enrolling an OCRA/QR token the sharedsecret=1 specifies, that you want to generate a shared secret
- activationcode - optional - if you are in Step1 of enrolling an OCRA token you need to pass the activation code, that was generated in the QRTAN-App

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`loadtokens()`

method: admin/loadtokens

description: loads a whole token file to the server

arguments:

- file - the file in a post request
- type - the file type.

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`losttoken()`

method: admin/losttoken

description: creates a new password token and copies the PIN and the user of the old token to the new token. The old token is disabled.

arguments:

- serial - serial of the old token

returns: a json result with the new serial an the password

exception: if an error occurs an exception is serialized and returned

`remove()`

method: admin/remove

description: deletes either a certain token given by serial or all tokens of a user

arguments:

- serial - optional
- user - optional

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`reset()`

method: admin/reset

description: reset the FailCounter of a Token

arguments: user or serial - to identify the tokens

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`resync()`

method: admin/resync - resync a token to a new counter

description: this function resync the token, if the counter on server side is out of sync with the physica token.

arguments:

- serial - serial or user required
- user - s.o.
- otp1 - the next otp to be found
- otp2 - the next otp after the otp1

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`set()`

method: admin/set

description: this function is used to set many different values of a token.

arguments:

- serial - optional
- user - optional
- pin - optional - set the OTP PIN
- MaxFailCount - optional - set the maximum fail counter of a token
- SyncWindow - optional - set the synchronization window of the token
- OtpLen - optional - set the OTP Lenght of the token
- CounterWindow - optional - set the counter window (blank presses)

- hashlib - optioanl - set the hashing algo for HMAC tokens. This can be sha1, sha256, sha512
- timeWindow - optional - set the synchronize window for timebased tokens (in seconds)
- timeStep - optional - set the timestep for timebased tokens (usually 30 or 60 seconds)
- timeShift - optional - set the shift or timedrift of this token
- countAuthSuccessMax - optional - set the maximum allowed successful authentications
- countAuthSuccess - optional - set the counter of the successful authentications
- countAuth - optional - set the counter of authentications
- countAuthMax - optional - set the maximum allowed authentication tries
- validityPeriodStart - optional - set the start date of the validity period. The token can not be used before this date
- validityPeriodEnd - optional - set the end date of the validaity period. The token can not be used after this date

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`setPin()`

method: admin/set

description: This function sets the smartcard PINs of a eTokenNG OTP. The userpin is used to store the mOTP PIN of mOTP tokens! !!! For setting the OTP PIN, use the function /admin/set!

arguments:

- serial - required
- userpin - optional: store the userpin
- sopin - optional: store the sopin

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`show()`

method: admin/show

description: displays the list of the available tokens

arguments:

- serial - optional: only this serial will be displayed
- user - optional: only the tokens of this user will be displayed. If the user does not exist, linotp will search tokens of users, who contain this substring. **TODO:** This can be very time consuming an will be changed in the next release to use wildcards.
- filter - optional: takes a substring to search in table token columns
- viewrealm - optional: takes a realm, only the tokens in this realm will be displayed
- sortby - optional: sort the output by column
- sortdir - optional: asc/desc

- page - optional: reqeuest a certain page
- pagesize- optional: limit the number of returned tokens

returns: a json result with: { “head”: [], “data”: [[row1], [row2] ..] }

exception: if an error occurs an exception is serialized and returned

`testresolver()`

method: admin/testresolver

description: This method tests a useridresolvers configuration

arguments:

- **type - “LDAP”: depending on the type there are other parameters:**
  - “SQL”
- **LDAP:**
  - BINDDN
  - BINDPW
  - LDAPURI
  - TIMEOUT
  - LDAPBASE
  - LOGINNAMEATTRIBUTE
  - LDAPSEARCHFILTER
  - LDAPFILTER
  - USERINFO
  - LDAPSEARCHFILTER
  - SIZELIMIT
  - NOREFERRALS
  - CACERTIFICATE
- **SQL:**
  - Driver
  - Server
  - Port
  - Database
  - User
  - Password
  - Table

returns:

a json result with a boolean “result”: true

exception: if an error occurs an exception is serialized and returned

`tokenrealm()`

method: admin/tokenrealm - set the realms a token belongs to

description: sets the realms of a token

arguments:

- serial - required - serialnumber of the token
- realms - required - comma seperated list of realms

`unassign()`

method: admin/unassign - remove the assigned user from the token

description: unassigns a token from a user. i.e. the binding between the token and the user is removed

arguments:

- serial - required - the serial number / identifier of the token
- user - optional

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`userlist()`

method: admin/userlist - list all users

description: lists the user in a realm

arguments:

- <searchexpr> - will be retrieved from the UserIdResolverClass
- realm - a realm, which is a collection of resolver configurations
- resConf - a destinct resolver configuration

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

2.3 System Controller

class `linotp.controllers.system.SystemController(`**args*, ***kw*`)`

The linotp.controllers are the implementation of the web-API to talk to the LinOTP server. The SystemController is used to configure the LinOTP server. The functions of the SystemController are invoked like this

https://server/system/<functionname>

The functions are described below in more detail.

`checkPolicy()`

method: system/checkPolicy

description: this function checks if a the given parameter will trigger a policy or not.

arguments:

- user - the name of the user
- realm - the realm
- scope - the scope
- action
- client - the client IP

returns:

a json result like this:

value [{ “allowed”][“true”,] “policy” : <Name der Policy, die das erlaubt hat> }

value [{ “allowed”][“false”,] “info” : <sowas wie die Fehlermeldung> }

`delConfig()`

method: system/delConfig

description: delete a configuration key

arguments: key - generic Configuration entry name

returns: a json result with the deleted value

exception: if an error occurs an exception is serializedsetConfig and returned

`delPolicy()`

method: system/delPolicy

description: this function deletes the policy with the given name

arguments: name - the policy with the given name

returns: a json result about the delete success

exception: if an error occurs an exception is serialized and returned

`delRealm()`

method: system/delRealm

description: this function deletes the given realm

arguments: realm - the name of the realm to be deleted

returns: a json result if deleting the realm was successful

exception: if an error occurs an exception is serialized and returned

`delResolver()`

method: system/delResolver

description: this function deletes an existing resolver All config keys of this resolver get deleted

arguments: resolver - the name of the resolver to delete.

returns: success state

exception: if an error occurs an exception is serialized and returned

`getConfig()`

method: system/getConfig

description: retrieve a configuration key. If no key is given, the complete configuration is returned

arguments: key - generic Configuration entry name

returns: a json result with the found value

exception: if an error occurs an exception is serialized and returned

`getDefaultRealm()`

method: system/getDefaultRealm

description: this function returns the default realm

arguments: ./.

returns: a json description of the default realm

exception: if an error occurs an exception is serialized and returned

`getPolicy()`

method: system/getPolicy

description: this function is used to retrieve the policies that you defined.

arguments:

- realm - (optional) will return all policies in the given realm
- name - (optional) will only return the policy with the given name
- scope - (optional) will only return the policies within the given scope
- **export - (optional) The filename needs to be specified as the third part of the URL like /system** will then be exported to this file.

returns: a json result with the configuration of the specified policies

exception: if an error occurs an exception is serialized and returned

`getPolicyDef()`

method: system/getPolicyDef

description: This is a helper function that returns the POSSIBLE policy definitions, that can be used to define your policies.

arguments: scope - optional - if given, the function will only return policy definitions for the given scope.

returns:

the policy definitions of

- allowed scopes
- allowed actions in scopes
- type of actions

exception: if an error occurs an exception is serialized and returned

`getRealms()`

method: system/getRealms

description: returns all realm definitinos as a json result.

arguments:

returns: a json result with a list of Realms

exception: if an error occurs an exception is serialized and returned

Either the admin has the policy scope=system, action=read or he is rights in scope=admin for some realms. If he does not have the system-read-right, then he will only see the realms, he is admin of.

`getResolver()`

method: system/getResolver

description: this function retrieves the definition of the resolver

arguments: resolver - the name of the resolver

returns: a json result with the configuration of a specified resolver

exception: if an error occurs an exception is serialized and returned

`getResolvers()`

method: system/getResolvers

descriptions: returns a json list of all useridresolvers

arguments:

returns: a json result with a list of all available resolvers

exception: if an error occurs an exception is serialized and returned

`importPolicy()`

method: system/importPolicy

description: This function is used to import policies from a file.

arguments: file - mandatory: The policy file in the POST request

`policies_flexi()`

This function is used to fill the policies tab Unlike the complex /system/getPolcies function, it only returns a simple array of the tokens.

`registerResolverClass()`

method: system/registerResolverClass

description: not yet implemented

arguments: name - the resolver class name

returns: a json result about the success

exception: if an error occurs an exception is serialized and returned

`setConfig()`

method: system/setConfig

description: set a configuration key

arguments: key - generic Configuration entry name value - generic Configuration value

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`setDefault()`

method: system/set

description: define default settings for tokens. These default settings are used when new tokens are generated. The default settings will not affect already enrolled tokens.

arguments: DefaultMaxFailCount - Default value for the maximum allowed authentication failures DefaultSyncWindow - Default value for the synchronization window DefaultCountWindow - Default value for the coutner window DefaultOtpLen - Default value for the OTP value length – usuall 6 or 8 DefaultResetFailCount - Default value, if the FailCounter should be reset on successful authentication [True|False]

returns:

a json result with a boolean "result": true

exception: if an error occurs an exception is serialized and returned

`setDefaultRealm()`

method: system/setDefaultRealm

description: this function sets the given realm to the default realm

arguments: realm - the name of the realm, that should be the default realm

returns: a json result with a list of Realms

exception: if an error occurs an exception is serialized and returned

`setPolicy()`

method: system/setPolicy

description: Stores a policy that define ACL or behaviour of several different actions in LinOTP. The policy is stored as configuration values like this:

Policy.<NAME>.action Policy.<NAME>.scope Policy.<NAME>.realm

arguments: name: name of the policy action: which action may be executed scope: selfservice realm: This polcy holds for this realm user: (optional) This polcy binds to this user time: (optional) on which time does this policy hold client: (optional) for which requesting client this should be

returns: a json result with success or error

exception: if an error occurs an exception is serialized and returned

`setRealm()`

method: system/setRealm

description: this function is used to define a realm with the given useridresolvers

arguments: realm - name of the realm resolvers - comma seperated list of resolvers, that should be

in this realm

returns: a json result with a list of Realms

exception: if an error occurs an exception is serialized and returned

`setResolver()`

method: system/setResolver

description: creates or updates a useridresolver

arguments: name - the name of the resolver type - the type of the resolver [ldapsersolver, sqlresolver]

LDAP: LDAPURI LDAPBASE BINDDN BINDPW TIMEOUT SIZELIMIT LOGINNAMEATTRIBUTE LDAPSEARCHFILTER LDAPFILTER USERINFO NOREFERRALS - True|False

SQL: Database Driver Server Port User Password Table Map

returns: a json result with the found value

exception: if an error occurs an exception is serialized and returned

2.4 License Controller

class `linotp.controllers.license.LicenseController(`**args*, ***kw*`)`

`getLicense()`
Returns the license information as a JSON object

`setLicense()`
sets the license.

param: license - containing the license file.

The method must be called as a POST request

2.5 Session protection

Starting with LinOTP 2.4.1 the API provides an additional session protection. This means all functions that change data need to pass an additional session parameter in the URL parameters like this:

https://linotpserver/admin/disable?serial=123456&session=ea49db685b8e7d2ff18ac50a57e9903630f0ed03e2cbe

If the session is not passed the server will respond with a 401 Not authenticated. To get such a session, the client needs to send a session request:

https://linotpserver/admin/getsession

This will set a cookie in the clients cookie store, which also needs to be passed with every request. Thus the clients needs to pass the cookie and the session parameter. The server will compare these two values and only perform the required action if the session values are the same.

2.5.1 Disabling session protection

You may disable session protection. Please note that you should not do this when accessing LinOTP with a web browser. You should only disable session protection if you fully understand what you are doing.

In `/etc/linotp2/linotp.ini` add the following line to the `DEFAULT` section:

```
linotpNoSessionCheck = 10.1.2.3, 192.168.0.0/24
```

You can add any IP address or subnet in CIDR notation seperated by comma.

3 | Tokens

If you decide to implement new token classes, you need to add the list of the Python modules containing the classes to the parameter `linotpTokenModules` in `/etc/linotp2/linotp.ini` in the `DEFAULT` section.

Note: The module `linotp.lib.tokenclass` will always be used.

Starting with LinOTP 2.5.2 it is even simpler to add new token types to LinOTP. The complete token type handling is now done dynamically. This means you can

- add a new token module
- new policies that are necerray for this token
- new management GUI (enrollment and token settings) and
- new selfservice enrollment GUI

for this new token without having to adapt anything in the LinOTP core.

This is done by creating two files

new_token.py

This file containes all the program logic, the calculation and verification of the OTP values and the handling of the policies.

new_token.mako

This file contains the GUI parts for management and selfservice.

The existing tokens are located in the module `linotp.lib.tokens`.

Both files describe the new token and need to be located in the same directory i.e. the same python module.

3.1 The GUI file

The **new_token.mako** file is written in the MAKO templating language. For more information on how MAKO works can be found on the project website [1].

Note: Please take a look at the example file `dyntoken.mako`.

To make the file handling easier there is only this single file providing the GUI for

- selfservice
- token configurations
- token enrollment.

[1] http://www.makotemplates.org/

Therefore there is a top level switch in this file that evaluates the variable `c.scope`.

3.1.1 The scope config

config

The scope `config` is used for the generic token configuration information. E.g. for a time based HMAC token (TOTP) you can use this scope to configure the default time window and the default OTP length. The scope `config` contains the complete HTML code of this config tab.

config.title

This is the header text of the config tab.

To read and write the token configuration to the LinOTP configuration database you need to define two javascript functions:

```
function dyn_get_config_val(){
       var ret = {};
   ret['DynTokenTimeOut']                    = 'dyn_challenge_timeout';
   ret['DynTokenMaxChallemges']      = 'dyn_max_challenge';
   return ret;
}

function dyn_get_config_params(){
   var ret ={}
   ret['DynTokenTimeOut']              = $('#dyn_challenge_timeout').val();
   ret['DynTokenMaxChallemges'] = $('#dyn_max_challenge').val();
   return ret
}
```

The function starts with the token type name of the token. You already know tokens like `hmac`, `totp`, `spass`. So the function names concatenate like `<token type>_get_config_val` and `<token type>_get_config_params()`. The example above are the functions of the token type `dyn`.

In the config tab for the token, you might have several HTML object. The functions map the identifier of the HTML objects to the configuration keys stored in the LinOTP configuration database.

The function `_get_config_val` is called, when the config tab is opened. It tells LinOTP to get the config entry with the key `DynTokenTimeOut` from the LinOTP configuration and put it into the HTML object with the `id=dyn_challenge_timeout`.

The function `_get_config_params` is called, when the config tab is saved. The values of the HTML objects are written to the corresponding LinOTP config database entries.

Warning: All config entries are stored in one config table. So you need to review third party token modules and assure, that a foreign token modules does not interfere with your other tokens or break your LinOTP. Best practice is to let your token config key start with the token type as prefix.

3.1.2 The scope enroll

enroll

This is the HTML GUI for the enrollment of a token by the administrator. Here all necessary data input are displayed.

enroll.title

This is the header text for the enrollment.

In the enrollment process an administrator needs to add additional information, like the RADIUS server in case of a RADIUS token, the secret key or whatever. These data need to be passed to LinOTP, therefore you need to define a javascript fuunction `<token type>_get_enroll_params`. The function has to return a hash of all the parameters, that will be passed to the `admin/init` function:

```
function dyn_get_enroll_params(){
   var url = {}

   url['type'] = 'dyn';
   url['otppin'] = $('#dyn_pin1').val();
   url['hashlib'] = $('#dyn_algorith').val();
   url['otplen'] = $('#dyn_otplen').val();
   url['serial'] = create_serial('DYN');

   if  ( $('#dyn_key_cb').attr('checked') ) {
      url['genkey']='1';}
   else {
      url['otpkey'] = $('#dyn_key').val();}

  return url;
}
```

3.1.3 The scope selfservice

selfservice.enroll

This is the HTML GUI for the user self enrollment in the selfservice portal.

selfservice.title.enroll

This is the header text of the selfservice tab.

In the self enrollment process several information needs to be entered by the user and needs to be passed to LinOTP. Three javascript functions are used to read the data from the HTML GUI and pass it to LinOTP. The first function `self_<token type>_get_param` function is used to retrieve the hashed parameters that needs to be passed to the `selfservice/userinit` function:

```
function self_dyn_get_param()
{
   var urlparam = {};

   urlparam['type']          = 'dyn';
   urlparam['description'] = "self enrolled" ;
   urlparam['otpkey']                = $('#motp_secret').val();
   urlparam['otppin']                = $('#motp_s_pin1').val();
   urlparam['serial']                = generate_serial("LSMO");

   return urlparam;
}
```

Note: The `generate_serial()` function generates a new serial number with the passed prefix.

The function `self_<token type>_clear` is used to clear the input fields:

```
function self_dyn_clear()
{
   $('#motp_s_pin1').val('');
   $('#motp_secret').val('');
   $('#motp_s_pin2').val('');
}
```

Finally there is a function `self_<token type>_submit` that is called when the token is enrolled. This function can also be used to do a input validation:

```
function self_dyn_submit(){
   var ret = false

   if ($('#form_registermotp').valid()) {
          var params =  self_motp_get_param();
              enroll_token( params );
              //self_motp_clear();
              ret = true;
   } else {
          alert("Form data not valid.");
   }
   return ret;
}
```

Note: When doing input validation the corresponding `jQuery.validator` needs to be defined.

3.2 The token file

The token file `dyntoken.py` itself contains the token definition as a python class that inherits from `TokenClass` which is defined in `linotp.lib.tokenclass`.

The token class needs to provide a class method `getClassInfo`, that tells LinOTP, which scopes to use in which situation. Therefore it returns a dictionary of the following kind:

```
{
    'type'           : 'dyn',
    'title'          : 'my dyn Token',
    'description'    : ('my own dynamic otp token'),

    'init'         : {'page' : {'html'      : 'dyntoken.mako',
                                'scope'     : 'enroll',},
                      'title'  : {'html'     : 'dyntoken.mako',
                                  'scope'    : 'enroll.tab',},
                         },

    'config'        : { 'page' : {'html'      : 'dyntoken.mako',
                                  'scope'     : 'config',},
                        'title'  : {'html'    : 'dyntoken.mako',
                                    'scope'    : 'config.tab',},
                      },

    'selfservice'   :  { 'enroll' : {'page' : {'html'  : 'dyntoken.mako',
                                               'scope' : 'selfservice.enroll',},
                                     'title' : { 'html' : 'dyntoken.mako',
                                                 'scope'  : 'selfservice.tab.enroll',},
                                        },
                         },


    }
```

3.2.1 Base TokenClass

class `linotp.lib.tokenclass.TokenClass`(*token*)

checkOtp(*anOtpVal1*, *counter*, *window*, *options=None*)
This checks the OTP value, AFTER the upper level did the checkPIN

return: counter of the matching OTP value.

checkPass(*passw*, *options=None*)
This function is called to ONLY check the OTP PIN of the token

Parameters

- **passw** – the to be checked pass (pin+otp)
- **options** – options an additional argument, which could be token specific

Returns True or False

Return type boolean

check_auth_counter()
This function checks the count_auth and the count_auth_success

check_otp_exist(*otp*, *window=None*)
checks if the given OTP value is/are values of this very token. This is used to autoassign and to determine the serial number of a token.

check_validity_period()
This checks if the datetime.datetime.now() is within the validity period of the token.

Returns either True/False

getInfo()
getInfo - return the status of the token rollout

Returns return the status dict.

Return type dict

getOtp(*curtTime=''*)
The default token does not support getting the otp value will return something like:

1, pin, otpval, combined

a negative value is a failure.

get_multi_otp(*count=0*, *epoch_start=0*, *epoch_end=0*, *curTime=None*)
This returns a dictionary of multiple future OTP values of a token.

parameter count - how many otp values should be returned epoch_start - time based tokens: start when epoch_end - time based tokens: stop when

return True/False error text OTP dictionary

get_validity_period_end()
returns the end of validity period (if set)

get_validity_period_start()
returns the start of validity period (if set)

incOtpCounter(*counter*, *reset=True*)

method incOtpCounter(aToken, counter)

parameters: token - a token object counter - the new counter reset - optional -

exception: in case of an transaction fail an exception is thrown

side effects: default of reset will reset the failCounter

setPin(*pin*, *param={}*)
set the PIN. The optional parameter “param” can hold the information, if the PIN is encrypted or hashed.

`setUid(`*uid*, *uidResolver*, *uidResClass*`)`
sets the UID values in the database

`setUser(`*user*, *report*`)`

Parameters

- **user** – a User() object, consisting of loginname and realm
- **report** – tbdf.

`set_count_auth(`*count*`)`
Sets the counter for the occurred login attepms

`set_count_auth_max(`*count*`)`
Sets the counter for the maximum allowed login attemps

`set_count_auth_success(`*count*`)`
Sets the counter for the occurred successfull logins

`set_count_auth_success_max(`*count*`)`
Sets the counter for the maximum allowed successfull logins

`set_validity_period_end(`*end_date*`)`
sets the end date of the validity period for a token

`set_validity_period_start(`*start_date*`)`
sets the start date of the validity period for a token

4 | UserIdResolver Modules

If you need to develop a new useridresolver that is to be loaded into LinOTP, this useridresolver needs to be a Python class, that inherits from the Python class UserIdResolver. Here you may find the minimum set of methods that needs to be implemented here:

http://linotp.org/api/useridresolverCE/useridresolver.UserIdResolver.UserIdResolver-class.html

For an detailed example, how these methods get implemented, you may consult the interface and implementation of the PasswdIdResolver:

http://linotp.org/api/useridresolverCE/useridresolver.PasswdIdResolver-module.html

Please note, that new UserIdResolvers need to comply with the AGPLv3 License and need also to be published under the AGPLv3 License.

class `useridresolver.UserIdResolver.UserIdResolver`

`checkPass(`*uid*, *password*`)`
: This function checks the password for a given uid. - returns true in case of success - false if password does not match

`getResolverId()`
: getResolverId(LoginName) - returns the resolver identifier string - empty string if not exist

`getUserId(`*loginName*`)`
: getUserId(LoginName) - returns the identifier string - empty string if not exist

`getUserInfo(`*UserID*`)`
: This function returns all user information for a given user object identified by UserID. Return value is a dictionary, if no object is found, the dictionary is empty

`getUserList(`*serachDict*`)`
: searchUserList(key, value) This function finds the user objects, that have the term 'value' in the user object field 'key' The key may be something like 'loginname' or 'email'. The value is a regular expression. Return value is a list of dictionaries (each dictionary contains a user object) or an empty string if no object is found.

`getUsername(`*userid*`)`
: getUsername(LoginId) - returns the loginname string - empty string if not exist

5 | Licenses

5.1 AGPLv3

GNU AFFERO GENERAL PUBLIC LICENSE

Version 3, 19 November 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU Affero General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only

if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

> a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
>
> b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
>
> c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
>
> d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

> a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
>
> b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
>
> c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
>
> d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
>
> e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

> a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
>
> b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
>
> c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
>
> d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
>
> e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
>
> f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published

by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.
>
> You should have received a copy of the GNU Affero General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If your software can interact with users remotely through a computer network, you should also make sure that it provides a way for users to get its source. For example, if your program is a web application, its interface could display a “Source” link that leads users to an archive of the code. There are many ways you could offer source, and different solutions will be better for different programs; see section 13 for the specific requirements.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU AGPL, see <http://www.gnu.org/licenses/>.

VI Indices and tables

- *genindex*
- *modindex*
- *search*

Index

A

AdminController (class in linotp.controllers.admin), 242
Administrators, 201
appadmin, 209
Appliance, 197, 219, 221, 225
Assign tokens, 27
assign() (linotp.controllers.admin.AdminController method), 242
authentication request, 36

B

Backup and restore, 221
Backup HSM, 153

C

calculateOtp() (linotp.controllers.ocra.OcraController method), 236
Certificate, 211
check() (linotp.controllers.validate.ValidateController method), 231
check_auth_counter() (linotp.lib.tokenclass.TokenClass method), 259
check_otp_exist() (linotp.lib.tokenclass.TokenClass method), 259
check_s() (linotp.controllers.validate.ValidateController method), 232
check_serial() (linotp.controllers.admin.AdminController method), 242
check_t() (linotp.controllers.ocra.OcraController method), 235
check_validity_period() (linotp.lib.tokenclass.TokenClass method), 259
checkOtp() (linotp.lib.tokenclass.TokenClass method), 258
checkPass() (linotp.lib.tokenclass.TokenClass method), 259
checkPass() (useridresolver.UserIdResolver.UserIdResolver method), 261
checkPolicy() (linotp.controllers.system.SystemController method), 248
checkstatus() (linotp.controllers.ocra.OcraController method), 235
Clickatell.com, 70
Client certificate authentication, 122
Configuration sets, 207
copyTokenPin() (linotp.controllers.admin.AdminController method), 242
copyTokenUser() (linotp.controllers.admin.AdminController method), 242
Count Window, 11
count window, 40

D

Database parameters, 144
Default Settings, 11
delConfig() (linotp.controllers.system.SystemController method), 249
delPolicy() (linotp.controllers.system.SystemController method), 249
delRealm() (linotp.controllers.system.SystemController method), 249
delResolver() (linotp.controllers.system.SystemController method), 249
Digest authentication, 122
disable() (linotp.controllers.admin.AdminController method), 243

E

emergency recovery, 225
enable() (linotp.controllers.admin.AdminController method), 243
Encryption key, 213

G

Get OTP values, 45
get_multi_otp() (linotp.lib.tokenclass.TokenClass method), 259
get_validity_period_end() (linotp.lib.tokenclass.TokenClass method), 259
get_validity_period_start() (linotp.lib.tokenclass.TokenClass method), 259
getActivationCode() (linotp.controllers.ocra.OcraController method), 236
getConfig() (linotp.controllers.system.SystemController method), 249
getDefaultRealm() (linotp.controllers.system.SystemController method), 249
getInfo() (linotp.lib.tokenclass.TokenClass method), 259

getLicense() (linotp.controllers.license.LicenseController method), 252
getOtp() (linotp.lib.tokenclass.TokenClass method), 259
getPolicy() (linotp.controllers.system.SystemController method), 250
getPolicyDef() (linotp.controllers.system.SystemController method), 250
getRealms() (linotp.controllers.system.SystemController method), 250
getResolver() (linotp.controllers.system.SystemController method), 250
getResolverId() (useridresolver.UserIdResolver.UserIdResolver method), 261
getResolvers() (linotp.controllers.system.SystemController method), 250
getSerialByOtp() (linotp.controllers.admin.AdminController method), 243
getsession() (linotp.controllers.admin.AdminController method), 243
getUserId() (useridresolver.UserIdResolver.UserIdResolver method), 261
getUserInfo() (useridresolver.UserIdResolver.UserIdResolver method), 261
getUserList() (useridresolver.UserIdResolver.UserIdResolver method), 261
getUsername() (useridresolver.UserIdResolver.UserIdResolver method), 261
Gnokii, 71

H

HA group, 154
hardware tokens, 21
HSM passwords, 156
HttpSMSProvider, 70

I

Implicit returning, 145
Import tokens, 21
importPolicy() (linotp.controllers.system.SystemController method), 251
Imprint for self service portal, 79
incOtpCounter() (linotp.lib.tokenclass.TokenClass method), 259
init() (linotp.controllers.admin.AdminController method), 243

L

LicenseController (class in linotp.controllers.license), 252
loadtokens() (linotp.controllers.admin.AdminController method), 244
Logging, 203
Lost token, 55
losttoken() (linotp.controllers.admin.AdminController method), 244

M

Microsoft SQL Server, 142
Migrating tokens, 36
Migration, 160
mOTP PIN, 32

N

Network flow, 102
Network integration, 102, 223

O

OCRA, 37
OcraController (class in linotp.controllers.ocra), 233
Orphaned tokens, 241
OTP Length, 11
Out of sync, 40

P

policies_flexi() (linotp.controllers.system.SystemController method), 251
policy checker, 60
Pool recycle, 145
Preseeded hardware tokens, 21

Q

QR-TAN, 37

R

RADIUS configuration, 205
RADIUS token, 36
Realm, 14
Redundancy, 215
registerResolverClass() (linotp.controllers.system.SystemController method), 251
Remote token, 36
remove() (linotp.controllers.admin.AdminController method), 245
request() (linotp.controllers.ocra.OcraController method), 233
reset() (linotp.controllers.admin.AdminController method), 245
Restore HSM, 154
Resync token, 40
resync() (linotp.controllers.admin.AdminController method), 245
Root user, 209
RSA, 160

S

SAML, 135
samlcheck() (linotp.controllers.validate.ValidateController method), 232

Searching users, 26
Security Officer Pin, 14
Service Temporarily Unavailable, 111
set() (linotp.controllers.admin.AdminController method), 245
set_count_auth() (linotp.lib.tokenclass.TokenClass method), 260
set_count_auth_max() (linotp.lib.tokenclass.TokenClass method), 260
set_count_auth_success() (linotp.lib.tokenclass.TokenClass method), 260
set_count_auth_success_max() (linotp.lib.tokenclass.TokenClass method), 260
set_validity_period_end() (linotp.lib.tokenclass.TokenClass method), 260
set_validity_period_start() (linotp.lib.tokenclass.TokenClass method), 260
setConfig() (linotp.controllers.system.SystemController method), 251
setDefault() (linotp.controllers.system.SystemController method), 251
setDefaultRealm() (linotp.controllers.system.SystemController method), 251
setLicense() (linotp.controllers.license.LicenseController method), 252
setPin() (linotp.controllers.admin.AdminController method), 246
setPin() (linotp.lib.tokenclass.TokenClass method), 259
setPolicy() (linotp.controllers.system.SystemController method), 252
setRealm() (linotp.controllers.system.SystemController method), 252
setResolver() (linotp.controllers.system.SystemController method), 252
setUid() (linotp.lib.tokenclass.TokenClass method), 259
setUser() (linotp.lib.tokenclass.TokenClass method), 260
shibboleth, 135
show() (linotp.controllers.admin.AdminController method), 246
simplecheck() (linotp.controllers.validate.ValidateController method), 232
smsout.de, 70
smspin() (linotp.controllers.validate.ValidateController method), 232
SMSProvider, 69
SMSProviderConfig, 69
SMSProviderTimeout, 69
ssh, 209
SSL Certificate, 211
Support file, 217
Sync Window, 11
SystemController (class in linotp.controllers.system), 248

T

testresolver() (linotp.controllers.admin.AdminController method), 247
Token administrators, 201
Token realm, 41
TokenClass (class in linotp.lib.tokenclass), 258
Tokeninfo, 41
tokenrealm() (linotp.controllers.admin.AdminController method), 247
Two factor authentication, 30

U

unassign() (linotp.controllers.admin.AdminController method), 248
Univention Corporate Server, 114
Updates, 219
UserIdResolver, 12
UserIdResolver (class in useridresolver.UserIdResolver), 261
userlist() (linotp.controllers.admin.AdminController method), 248

V

ValidateController (class in linotp.controllers.validate), 231
Validity period, 41
Vasco Digipass, 25
Virtual Appliance, 101

Y

Yubikey components, 127