Postfix SASL Howto

Warning

People who go to the trouble of installing Postfix may have the expectation that Postfix is more secure than some other mailers. The Cyrus SASL library contains a lot of code. With this, Postfix becomes as secure as other mail systems that use the Cyrus SASL library. Dovecot provides an alternative that may be worth considering.

How Postfix uses SASL authentication

SMTP servers need to decide whether an SMTP client is authorized to send mail to remote destinations, or only to destinations that the server itself is responsible for. Usually, SMTP servers accept mail to remote destinations when the client's IP address is in the "same network" as the server's IP address.

SMTP clients outside the SMTP server's network need a different way to get "same network" privileges. To address this need, Postfix supports SASL authentication (RFC 4954[1], formerly RFC 2554[2]). With this a remote SMTP client can authenticate to the Postfix SMTP server, and the Postfix SMTP client can authenticate to a remote SMTP server. Once a client is authenticated, a server can give it "same network" privileges.

Postfix does not implement SASL itself, but instead uses existing implementations as building blocks. This means that some SASL-related configuration files will belong to Postfix, while other configuration files belong to the specific SASL implementation that Postfix will use. This document covers both the Postfix and non-Postfix configuration.

You can read more about the following topics:

Configuring SASL authentication in the Postfix SMTP server

As mentioned earlier, SASL is implemented separately from Postfix. For this reason, configuring SASL authentication in the Postfix SMTP server involves two different steps:

- Configuring the SASL implementation to offer a list of mechanisms that are suitable for SASL authentication and, depending on the SASL implementation used, configuring authentication backends that verify the remote SMTP client's authentication data against the system password file or some other database.
- Configuring the Postfix SMTP server to enable SASL authentication, and to authorize clients to relay mail or to control what envelope sender addresses the client may use.

Successful authentication in the Postfix SMTP server requires a functional SASL framework. Configuring SASL should therefore always be the first step, before configuring Postfix.

You can read more about the following topics:

Which SASL Implementations are supported?

Currently the Postfix SMTP server supports the Cyrus SASL and Dovecot SASL implementations.

> ***Note***
> *Current Postfix versions have a plug-in architecture that can support multiple SASL implementations. Before Postfix version 2.3, Postfix had support only for Cyrus SASL.*

To find out what SASL implementations are compiled into Postfix, use the following commands:

```
% postconf -a (SASL support in the SMTP server)
% postconf -A (SASL support in the SMTP+LMTP client)
```

These commands are available only with Postfix version 2.3 and later.

Configuring Dovecot SASL

Dovecot is a POP/IMAP server that has its own configuration to authenticate POP/IMAP clients. When the Postfix SMTP server uses Dovecot SASL, it reuses parts of this configuration. Consult the Dovecot documentation[3] for how to configure and operate the Dovecot authentication server.

Postfix to Dovecot SASL communication

Communication between the Postfix SMTP server and Dovecot SASL happens over a UNIX-domain socket or over a TCP socket. We will be using a UNIX-domain socket for better privacy.

The following fragment for Dovecot version 2 assumes that the Postfix queue is under `/var/spool/postfix/`.

```
1  conf.d/10-master.conf:
2      service auth {
3        ...
4        unix_listener /var/spool/postfix/private/auth {
5          mode = 0660
6          # Assuming the default Postfix user and group
7          user = postfix
8          group = postfix
9        }
10       ...
11     }
12
13 conf.d/10-auth.conf
14     auth_mechanisms = plain login
```

Line 4 places the Dovecot SASL socket in `/var/spool/postfix/private/auth`, lines 5-8 limit read+write permissions to user and group `postfix` only, and line 14 provides `plain` and `login` as mechanisms for the Postfix SMTP server.

Proceed with the section "Enabling SASL authentication and authorization in the

Postfix SMTP server" to turn on and use SASL in the Postfix SMTP server.

Configuring Cyrus SASL

The Cyrus SASL framework supports a wide variety of applications (POP, IMAP, SMTP, etc.). Different applications may require different configurations. As a consequence each application may have its own configuration file.

The first step configuring Cyrus SASL is to determine name and location of a configuration file that describes how the Postfix SMTP server will use the SASL framework.

Cyrus SASL configuration file name

The name of the configuration file (default: `smtpd.conf`) is configurable. It is a concatenation from a value that the Postfix SMTP server sends to the Cyrus SASL library, and the suffix `.conf`, added by Cyrus SASL.

The value sent by Postfix is the name of the server component that will use Cyrus SASL. It defaults to `smtpd` and is configured with one of the following variables:

> *`/etc/postfix/main.cf`[4]:*
>
> ```
> # Postfix 2.3 and later
> smtpd_sasl_path = smtpd
>
> # Postfix < 2.3
> smtpd_sasl_application_name = smtpd
> ```

Cyrus SASL configuration file location

The location where Cyrus SASL searches for the named file depends on the Cyrus SASL version and the OS/distribution used.

You can read more about the following topics:

- Cyrus SASL version 2.x searches for the configuration file in `/usr/lib/sasl2/`.
- Cyrus SASL version 2.1.22 and newer additionally search in `/etc/sasl2/`.
- Some Postfix distributions are modified and look for the Cyrus SASL configuration file in `/etc/postfix/sasl/`, `/var/lib/sasl2/` etc. See the distribution-specific documentation to determine the expected location.

> ***Note***
> *Cyrus SASL searches `/usr/lib/sasl2/` first. If it finds the specified configuration file there, it will not examine other locations.*

Postfix to Cyrus SASL communication

As the Postfix SMTP server is linked with the Cyrus SASL library `libsasl`, communication between Postfix and Cyrus SASL takes place by calling functions in the SASL library.

The SASL library may use an external password verification service, or an internal plugin to connect to authentication backends and verify the SMTP client's authentication data against the system password file or other databases.

The following table shows typical combinations discussed in this document:

authentication backend	*password verification service / plugin*
/etc/shadow	***saslauthd***
PAM	***saslauthd***
IMAP server	***saslauthd***
sasldb	***sasldb***
MySQL, PostgreSQL, SQLite	***sql***
LDAP	***ldapdb***

> ***Note***
> *Read the Cyrus SASL documentation for other backends it can use.*

saslauthd - Cyrus SASL password verification service

Communication between the Postfix SMTP server (read: Cyrus SASL's `libsasl`) and the `saslauthd` server takes place over a UNIX-domain socket.

`saslauthd` usually establishes the UNIX domain socket in `/var/run/saslauthd/` and waits for authentication requests. The Postfix SMTP server must have read+execute permission to this directory or authentication attempts will fail.

> ***Important***
> *Some distributions require the user `postfix` to be member of a special group e.g. `sasl`, otherwise it will not be able to access the `saslauthd` socket directory.*

The following example configures the Cyrus SASL library to contact `saslauthd` as its password verification service:

```
/etc/sasl2/smtpd.conf:
    pwcheck_method: saslauthd
    mech_list: PLAIN LOGIN
```

> ***Important***
> *Do not specify any other mechanisms in `mech_list` than `PLAIN` or `LOGIN` when using `saslauthd`! It can only handle these two mechanisms, and authentication will fail if clients are allowed to choose other mechanisms.*

> ***Important***
> *Plaintext mechanisms (`PLAIN`, `LOGIN`) send credentials unencrypted. This information should be protected by an additional security layer such as a TLS-encrypted SMTP session (see: TLS_README[5]).*

Additionally the `saslauthd` server itself must be configured. It must be told which authentication backend to turn to for password verification. The backend is selected with a `saslauthd` command-line option and will be shown in the following examples.

> ***Note***
> *Some distributions use a configuration file to provide saslauthd command line*

> *options to set e.g. the authentication backend. Typical locations are `/etc/sysconfig/saslauthd` or `/etc/default/saslauthd`.*

Using saslauthd with /etc/shadow

Access to the `/etc/shadow` system password file requires `root` privileges. The Postfix SMTP server (and in consequence `libsasl` linked to the server) runs with the least privilege possible. Direct access to `/etc/shadow` would not be possible without breaking the Postfix security architecture.

The `saslauthd` socket builds a safe bridge. Postfix, running as limited user `postfix`, can access the UNIX-domain socket that `saslauthd` receives commands on; `saslauthd`, running as privileged user `root`, has the privileges required to access the shadow file.

The `saslauthd` server verifies passwords against the authentication backend `/etc/shadow` if started like this:

```
% saslauthd -a shadow
```

See section "Testing saslauthd authentication" for test instructions.

Using saslauthd with PAM

Cyrus SASL can use the PAM framework to authenticate credentials. `saslauthd` uses the PAM framework when started like this:

```
% saslauthd -a pam
```

> ***Note***
> *PAM configuration for the Postfix SMTP server is usually given in `/etc/pam.d/smtp` and is beyond the scope of this document.*

See section "Testing saslauthd authentication" for test instructions.

Using saslauthd with an IMAP server

`saslauthd` can verify the SMTP client credentials by using them to log into an IMAP server. If the login succeeds, SASL authentication also succeeds. `saslauthd` contacts an IMAP server when started like this:

```
% saslauthd -a rimap -O imap.example.com
```

> ***Note***
> *The option "`-O imap.example.com`" specifies the IMAP server `saslauthd` should contact when it verifies credentials.*

> ***Important***
> *`saslauthd` sends IMAP login information unencrypted. Any IMAP session leaving the local host should be protected by an additional security layer such as an SSL tunnel.*

See section "Testing saslauthd authentication" for test instructions.

Testing saslauthd authentication

Cyrus SASL provides the `testsaslauthd` utility to test `saslauthd` authentication. The username and password are given as command line arguments. The example shows the response when authentication is successful:

```
% testsaslauthd -u username -p password
0: OK "Success."
```

> ***Note***
> *Sometimes the `testsaslauthd` program is not distributed with a the Cyrus SASL main package. In that case, it may be distributed with `-devel`, `-dev` or `-debug` packages.*

Specify an additional "`-s smtp`" if `saslauthd` was configured to contact the PAM authentication framework, and specify an additional "`-f /path/to/socketdir/mux`" if `saslauthd` establishes the UNIX-domain socket in a non-default location.

If authentication succeeds, proceed with the section "Enabling SASL authentication and authorization in the Postfix SMTP server".

Cyrus SASL Plugins - auxiliary property plugins

Cyrus SASL uses a plugin infrastructure (called `auxprop`) to expand `libsasl`'s capabilities. Currently Cyrus SASL sources provide three authentication plugins.

> *Plugin* *Description*
>
> ***sasldb*** *Accounts are stored stored in a Cyrus SASL Berkeley DB database*
>
> ***sql*** *Accounts are stored in a SQL database*
>
> ***ldapdb*** *Accounts are stored stored in an LDAP database*

> ***Important***
> *These three plugins support shared-secret mechanisms i.e. CRAM-MD5, DIGEST-MD5 and NTLM. These mechanisms send credentials encrypted but their verification process requires the password to be available in plaintext. Consequently passwords cannot (!) be stored in encrypted form.*

The sasldb plugin

The sasldb auxprop plugin authenticates SASL clients against credentials that are stored in a Berkeley DB database. The database schema is specific to Cyrus SASL. The database is usually located at `/etc/sasldb2`.

> ***Note***
> *The `sasldb2` file contains passwords in plaintext, and should have read+write access only to user `postfix` or a group that `postfix` is member of.*

The `saslpasswd2` command-line utility creates and maintains the database:

```
% saslpasswd2 -c -u example.com username
Password:
Again (for verification):
```

This command creates an account *username@example.com*.

> ***Important***
> *users must specify username@example.com as login name, not username.*

Run the following command to reuse the Postfix `mydomain` parameter value as the login domain:

```
% saslpasswd2 -c -u `postconf -h mydomain` username
Password:
Again (for verification):
```

> ***Note***
> *Run saslpasswd2 without any options for further help on how to use the command.*

The `sasldblistusers2` command lists all existing users in the sasldb database:

```
% sasldblistusers2
username1@example.com: password1
username2@example.com: password2
```

Configure libsasl to use sasldb with the following instructions:

```
/etc/sasl2/smtpd.conf:
    pwcheck_method: auxprop
    auxprop_plugin: sasldb
    mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5 NTLM
```

> ***Note***
> *In the above example adjust mech_list to the mechanisms that are applicable for your environment.*

The sql plugin

The sql auxprop plugin is a generic SQL plugin. It provides access to credentials stored in a MySQL, PostgreSQL or SQLite database. This plugin requires that SASL client

passwords are stored as plaintext.

> ***Tip***
> *If you must store encrypted passwords, you cannot use the sql auxprop plugin. Instead, see section "Using saslauthd with PAM", and configure PAM to look up the encrypted passwords with, for example, the `pam_mysql` module. You will not be able to use any of the methods that require access to plaintext passwords, such as the shared-secret methods CRAM-MD5 and DIGEST-MD5.*

The following example configures libsasl to use the sql plugin and connects it to a PostgreSQL server:

```
/etc/sasl2/smtpd.conf:
    pwcheck_method: auxprop
    auxprop_plugin: sql
    mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5 NTLM
    sql_engine: pgsql
    sql_hostnames: 127.0.0.1, 192.0.2.1
    sql_user: username
    sql_passwd: secret
    sql_database: dbname
    sql_select: SELECT password FROM users WHERE user = '%u@%r'
```

> ***Note***
> *Set appropriate permissions if `smtpd.conf` contains a password. The file should be readable by the `postfix` user.*

> ***Note***
> *In the above example, adjust `mech_list` to the mechanisms that are applicable for your environment.*

The sql plugin has the following configuration options:

> *sql_engine*
> *Specify `mysql` to connect to a MySQL server, `pgsql` for a PostgreSQL server or `sqlite` for an SQLite database*
> *sql_hostnames*

> *Specify one or more servers (hostname or hostname:port) separated by commas.*
>
> > ***Note***
> > *With MySQL servers, specify `localhost` to connect over a UNIX-domain socket, and specify `127.0.0.1` to connect over a TCP socket.*
>
> *sql_user*
> *The login name to gain access to the database.*
> *sql_passwd*
> *The password to gain access to the database.*
> *sql_database*
> *The name of the database to connect to.*
> *sql_select*
> *The SELECT statement that should retrieve the plaintext password from a database table.*
>
> > ***Important***
> > *Do not enclose the statement in quotes! Use single quotes to escape macros!*

The sql plugin provides macros to build `sql_select` statements. They will be replaced with arguments sent from the client. The following macros are available:

> *%u*
> *The name of the user whose properties are being selected.*
> *%p*
> *The name of the property being selected. While this could technically be anything, Cyrus SASL will try userPassword and cmusaslsecretMECHNAME (where MECHNAME is the name of a SASL mechanism).*
> *%r*
> *The name of the realm to which the user belongs. This could be the KERBEROS realm, the fully-qualified domain name of the computer the SASL application is running on, or the domain after the "@" in a username.*

The ldapdb plugin

The ldapdb auxprop plugin provides access to credentials stored in an LDAP server. This plugin requires that SASL client passwords are stored as plaintext.

> ***Tip***
> *If you must store encrypted passwords, you cannot use the ldapdb auxprop plugin. Instead, you can use "`saslauthd -a ldap`" to query the LDAP database directly, with appropriate configuration in `saslauthd.conf`, as described here[6]. You will not be able to use any of the methods that require access to plaintext passwords, such as the shared-secret methods CRAM-MD5 and DIGEST-MD5.*

The ldapdb plugin implements proxy authorization. This means that the ldapdb plugin uses its own username and password to authenticate with the LDAP server, before it asks the LDAP server for the remote SMTP client's password. The LDAP server then decides if the ldapdb plugin is authorized to read the remote SMTP client's password.

In a nutshell: Configuring ldapdb means authentication and authorization must be configured twice - once in the Postfix SMTP server to authenticate and authorize the remote SMTP client, and once in the LDAP server to authenticate and authorize the ldapdb plugin.

This example configures libsasl to use the ldapdb plugin and the plugin to connect to an LDAP server:

```
/etc/sasl2/smtpd.conf:
    pwcheck_method: auxprop
    auxprop_plugin: ldapdb
    mech_list: PLAIN LOGIN NTLM CRAM-MD5 DIGEST-MD5
    ldapdb_uri: ldap[7]://localhost
    ldapdb_id: proxyuser
    ldapdb_pw: password
    ldapdb_mech: DIGEST-MD5
```

> ***Important***
> *Set appropriate permissions if `smtpd.conf` contains a password. The file should be readable by the `postfix` user.*

> ***Note***
> *The shared-secret mechanisms (CRAM-MD5, etc.) require that the SASL client passwords are stored as plaintext.*

The following is a summary of applicable `smtpd.conf` file entries:

> *auxprop_plugin*
> *Specify `ldapdb` to enable the plugin.*
> *ldapdb_uri*
> *Specify either `ldapi://` for to connect over a UNIX-domain socket, `ldap`[8]`://` for an unencrypted TCP connection or `ldaps://` for an encrypted TCP connection.*
> *ldapdb_id*
> *The login name to authenticate the ldapdb plugin to the LDAP server (proxy authorization).*
> *ldapdb_pw*
> *The password (in plaintext) to authenticate the ldapdb plugin to the LDAP server (proxy authorization).*
> *ldapdb_mech*
> *The mechanism to authenticate the ldapdb plugin to the LDAP server.*
>
> > ***Note***
> > *Specify a mechanism here that is supported by the LDAP server.*
>
> *ldapdb_rc (optional)*
> *The path to a file containing individual configuration options for the ldapdb LDAP client (libldap). This allows to specify a TLS client certificate which in turn can be used to use the SASL EXTERNAL mechanism.*
>
> > ***Note***
> > *This mechanism supports authentication over an encrypted transport layer, which is recommended if the plugin must connect to an OpenLDAP server on a remote machine.*
>
> *ldapdb_starttls (optional)*
> *The TLS policy for connecting to the LDAP server. Specify either `try` or `demand`. If the option is `try` the plugin will attempt to establish a TLS-encrypted connection with the LDAP server, and will fallback to an unencrypted connection if TLS fails. If the policy is `demand` and a TLS-encrypted connection cannot be established, the connection fails immediately.*

When the ldapdb plugin connects to the OpenLDAP server and successfully authenticates, the OpenLDAP server decides if the plugin user is authorized to read SASL account information.

The following configuration gives an example of authorization configuration in the OpenLDAP slapd server:

```
/etc/openldap/slapd.conf:
    authz-regexp
    uid=(.*),cn=.*,cn=auth
    ldap[9]:///dc=example,dc=com??sub?cn=$1
    authz-policy to
```

Here, the `authz-regexp` option serves for authentication of the ldapdb user. It maps its login name to a DN in the LDAP directory tree where `slapd` can look up the SASL account information. The `authz-policy` options defines the authentication policy. In this case it grants authentication privileges "`to`" the ldapdb plugin.

The last configuration step is to tell the OpenLDAP `slapd` server where ldapdb may search for usernames matching the one given by the mail client. The example below adds an additional attribute ldapdb user object (here: `authzTo` because the authz-policy is "`to`") and configures the scope where the login name "proxyuser" may search:

```
dn: cn=proxyuser,dc=example,dc=com
changetype: modify
add: authzTo
authzTo: dn.regex:uniqueIdentifier=(.*),ou=people,dc=example,dc=com
```

Use the `ldapmodify` or `ldapadd` command to add the above attribute.

> ***Note***
>
> *Read the chapter "Using SASL" in the OpenLDAP Admin Guide[10] for more detailed instructions to set up SASL authentication in OpenLDAP.*

Enabling SASL authentication and authorization in the Postfix SMTP server

By default the Postfix SMTP server uses the Cyrus SASL implementation. If the Dovecot SASL implementation should be used, specify an `smtpd_sasl_type` value of `dovecot` instead of `cyrus`:

> */etc/postfix/main.cf*[11]*:*
>
> *smtpd_sasl_type = dovecot*

Additionally specify how Postfix SMTP server can find the Dovecot authentication server. This depends on the settings that you have selected in the section "Postfix to Dovecot SASL communication".

- If you configured Dovecot for UNIX-domain socket communication, configure Postfix as follows:

  `/etc/postfix/main.cf`[12]`:`

  ```
  smtpd_sasl_path = private/auth
  ```

  Note
  This example uses a pathname relative to the Postfix queue directory, so that it will work whether or not the Postfix SMTP server runs chrooted.
- If you configured Dovecot for TCP socket communication, configure Postfix as follows. If Dovecot runs on a different machine, replace 127.0.0.1 by that machine's IP address.

  `/etc/postfix/main.cf`[13]`:`

  ```
  smtpd_sasl_path = inet:127.0.0.1:12345
  ```

  Note
  If you specify a remote IP address, information will be sent as plaintext over the network.

Enabling SASL authentication in the Postfix SMTP server

Regardless of the SASL implementation type, enabling SMTP authentication in the Postfix SMTP server always requires setting the `smtpd_sasl_auth_enable` option:

> */etc/postfix/main.cf*[14]*:*
>
> *smtpd_sasl_auth_enable = yes*

After a "postfix reload", SMTP clients will see the additional capability AUTH in an SMTP session, followed by a list of authentication mechanisms the server supports:

```
% telnet server.example.com 25
...
220 server.example.com ESMTP Postfix
EHLO client.example.com
250-server.example.com
250-PIPELINING
250-SIZE 10240000
250-AUTH DIGEST-MD5 PLAIN CRAM-MD5
...
```

However not all clients recognize the AUTH capability as defined by the SASL authentication RFC. Some historical implementations expect the server to send an "=" as separator between the AUTH verb and the list of mechanisms that follows it.

The `broken_sasl_auth_clients` configuration option lets Postfix repeat the AUTH statement in a form that these broken clients understand:

```
/etc/postfix/main.cf[15]:
    broken_sasl_auth_clients = yes
```

> ***Note***
> *Enable this option for Outlook up to and including version 2003 and Outlook Express up to version 6. This option does not hurt other clients.*

After "postfix reload", the Postfix SMTP server will propagate the AUTH capability twice - once for compliant and once for broken clients:

```
% telnet server.example.com 25
...
220 server.example.com ESMTP Postfix
EHLO client.example.com
250-server.example.com
250-PIPELINING
250-SIZE 10240000
250-AUTH DIGEST-MD5 PLAIN CRAM-MD5
250-AUTH=DIGEST-MD5 PLAIN CRAM-MD5
...
```

Postfix SMTP Server policy - SASL mechanism properties

The Postfix SMTP server supports policies that limit the SASL mechanisms that it makes available to clients, based on the properties of those mechanisms. The next two sections give examples of how these policies are used.

> | *Property* | *Description* |
> | --- | --- |
> | *noanonymous* | *Don't use mechanisms that permit anonymous authentication.* |
> | *noplaintext* | *Don't use mechanisms that transmit unencrypted username and password information.* |
> | *nodictionary* | *Don't use mechanisms that are vulnerable to dictionary attacks.* |
> | *forward_secrecy* | *Require forward secrecy between sessions (breaking one session does not break earlier sessions).* |
> | *mutual_auth* | *Use only mechanisms that authenticate both the client and the server to each other.* |

Unencrypted SMTP session

The default policy is to allow any mechanism in the Postfix SMTP server except for those based on anonymous authentication:

> *`/etc/postfix/main.cf`[16]:*
>
> ```
> # Specify a list of properties separated by comma or whitespace
> smtpd_sasl_security_options = noanonymous
> ```

> ***Important***
> *Always set at least the `noanonymous` option. Otherwise, the Postfix SMTP server can give strangers the same authorization as a properly-authenticated client.*

Encrypted SMTP session (TLS)

A separate parameter controls Postfix SASL mechanism policy during a TLS-encrypted SMTP session. The default is to copy the settings from the unencrypted session:

```
/etc/postfix/main.cf[17]:
    smtpd_sasl_tls_security_options = $smtpd_sasl_security_options
```

A more sophisticated policy allows plaintext mechanisms, but only over a TLS-encrypted connection:

```
/etc/postfix/main.cf[18]:
    smtpd_sasl_security_options = noanonymous, noplaintext
    smtpd_sasl_tls_security_options = noanonymous
```

To offer SASL authentication only after a TLS-encrypted session has been established specify this:

```
/etc/postfix/main.cf[19]:
    smtpd_tls_auth_only = yes
```

Enabling SASL authorization in the Postfix SMTP server

After the client has authenticated with SASL, the Postfix SMTP server decides what the remote SMTP client will be authorized for. Examples of possible SMTP clients authorizations are:

- Send a message to a remote recipient.
- Use a specific envelope sender in the MAIL FROM command.

These permissions are not enabled by default.

Mail relay authorization

With `permit_sasl_authenticated` the Postfix SMTP server can allow SASL-authenticated SMTP clients to send mail to remote destinations. Examples:

```
# With Postfix 2.10 and later, the mail relay policy is
# preferably specified under smtpd_relay_restrictions.
/etc/postfix/main.cf[20]:
```

```
    smtpd_relay_restrictions =
        permit_mynetworks
        permit_sasl_authenticated
        reject_unauth_destination
# Older configurations combine relay control and spam control under
# smtpd_recipient_restrictions. To use this example with Postfix ≥
# 2.10 specify "smtpd_relay_restrictions=".
/etc/postfix/main.cf[21]:
    smtpd_recipient_restrictions =
        permit_mynetworks
        permit_sasl_authenticated
        reject_unauth_destination
        ...other rules...
```

Envelope sender address authorization

By default an SMTP client may specify any envelope sender address in the MAIL FROM command. That is because the Postfix SMTP server only knows the remote SMTP client hostname and IP address, but not the user who controls the remote SMTP client.

This changes the moment an SMTP client uses SASL authentication. Now, the Postfix SMTP server knows who the sender is. Given a table of envelope sender addresses and SASL login names, the Postfix SMTP server can decide if the SASL authenticated client is allowed to use a particular envelope sender address:

```
/etc/postfix/main.cf[22]:
    smtpd_sender_login_maps = hash:/etc/postfix/controlled_envelope_senders

    smtpd_recipient_restrictions =
        ...
        reject_sender_login_mismatch
        permit_sasl_authenticated
        ...
```

The `controlled_envelope_senders` table specifies the binding between a sender envelope address and the SASL login names that own that address:

```
/etc/postfix/controlled_envelope_senders
    # envelope sender           owners (SASL login names)
    john@example.com            john@example.com
    helpdesk@example.com        john@example.com, mary@example.com
    postmaster                  admin@example.com
    @example.net                barney, fred, john@example.com, mary@example.com
```

With this, the `reject_sender_login_mismatch` restriction above will reject the sender address in the MAIL FROM command if `smtpd_sender_login_maps` does not specify the SMTP client's login name as an owner of that address.

See also `reject_authenticated_sender_login_mismatch`, `reject_known_sender_login_mismatch`, and `reject_unauthenticated_sender_login_mismatch` for additional control over the SASL login name and the envelope sender.

Additional SMTP Server SASL options

Postfix provides a wide range of SASL authentication configuration options. The next section lists a few that are discussed frequently. See postconf(5)[23] for a complete list.

Per-account access control

Postfix can implement policies that depend on the SASL login name (Postfix 2.11 and later). Typically this is used to HOLD or REJECT mail from accounts whose credentials have been compromised.

```
/etc/postfix/main.cf[24]:
    smtpd_recipient_restrictions =
        permit_mynetworks
        check_sasl_access hash:/etc/postfix/sasl_access
        permit_sasl_authenticated
        ...

/etc/postfix/sasl_access:
     # Use this when smtpd_sasl_local_domain is empty.
```

```
username   HOLD
# Use this when smtpd_sasl_local_domain=example.com.
username@example.com HOLD
```

Default authentication domain

Postfix can append a domain name (or any other string) to a SASL login name that does not have a domain part, e.g. "`john`" instead of "`john@example.com`":

```
/etc/postfix/main.cf[25]:
    smtpd_sasl_local_domain = example.com
```

This is useful as a default setting and safety net for misconfigured clients, or during a migration to an authentication method/backend that requires an authentication REALM or domain name, before all SMTP clients are configured to send such information.

Hiding SASL authentication from clients or networks

Some clients insist on using SASL authentication if it is offered, even when they are not configured to send credentials - and therefore they will always fail and disconnect.

Postfix can hide the AUTH capability from these clients/networks:

```
/etc/postfix/main.cf[26]:
    smtpd_sasl_exceptions_networks = !192.0.2.171/32, 192.0.2.0/24
```

Adding the SASL login name to mail headers

To report SASL login names in Received: message headers (Postfix version 2.3 and later):

```
/etc/postfix/main.cf[27]:
    smtpd_sasl_authenticated_header = yes
```

> ***Note***
> *The SASL login names will be shared with the entire world.*

Testing SASL authentication in the Postfix SMTP Server

To test the server side, connect (for example, with `telnet`) to the Postfix SMTP server port and you should be able to have a conversation as shown below. Information sent by the client (that is, you) is shown in **bold** font.

```
% telnet server.example.com 25
...
220 server.example.com ESMTP Postfix
EHLO client.example.com
250-server.example.com
250-PIPELINING
250-SIZE 10240000
250-ETRN
250-AUTH DIGEST-MD5 PLAIN CRAM-MD5
250 8BITMIME
AUTH PLAIN AHRlc3QAdGVzdHBhc3M=
235 Authentication successful
```

To test this over a connection that is encrypted with TLS, use `openssl s_client` instead of `telnet`:

```
% openssl s_client -connect server.example.com:25 -starttls smtp
...
220 server.example.com ESMTP Postfix
EHLO client.example.com
...see above example for more...
```

Instead of `AHRlc3QAdGVzdHBhc3M=`, specify the base64-encoded form of `\0username\0password` (the \0 is a null byte). The example above is for a user named `` `test' `` with password `` `testpass' ``.

> ***Caution***
> *When posting logs of the SASL negotiations to public lists, please keep in mind that*

> *username/password information is trivial to recover from the base64-encoded form.*

You can use one of the following commands to generate base64 encoded authentication information:

- Using a recent version of the **bash** shell:

  ```
  % echo -ne '\000username\000password' | openssl base64
  ```

  Some other shells support similar syntax.
- Using the **printf** command:

  ```
  % printf '\0%s\0%s' 'username' 'password' | openssl base64
  % printf '\0%s\0%s' 'username' 'password' | mmencode
  ```

  The **mmencode** command is part of the metamail software.
- Using Perl **MIME::Base64** (from http://www.cpan.org/[28]):

  ```
  % perl -MMIME::Base64 -e \
      'print encode_base64("\0username\0password");'
  ```

  If the username or password contain "@", you must specify "\@".
- Using the **gen-auth** script:

  ```
  % gen-auth plain
  username: username
  password:
  ```

  The **gen-auth** Perl script was written by John Jetmore and can be found at http://jetmore.org/john/code/gen-auth[29].

Configuring SASL authentication in the Postfix SMTP/LMTP client

The Postfix SMTP and the LMTP client can authenticate with a remote SMTP server via the Cyrus SASL framework. At this time, the Dovecot SASL implementation does not provide client functionality.

> ***Note***
> *The examples in this section discuss only the SMTP client. Replace `smtp_` with `lmtp_` to get the corresponding LMTP client configuration.*

You can read more about the following topics:

Enabling SASL authentication in the Postfix SMTP/LMTP client

This section shows a typical scenario where the Postfix SMTP client sends all messages via a mail gateway server that requires SASL authentication.

> ***Trouble solving tips:***
> - *If your SASL logins fail with "SASL authentication failure: No worthy mechs found" in the mail logfile, then see the section "**Postfix SMTP/LMTP client policy - SASL mechanism properties**".*
> - *For a solution to a more obscure class of SASL authentication failures, see "**Postfix SMTP/LMTP client policy - SASL mechanism names**".*

To make the example more readable we introduce it in two parts. The first part takes care of the basic configuration, while the second part sets up the username/password information.

> */etc/postfix/**main.cf**[30]:*
>
> ```
> smtp_sasl_auth_enable = yes
> relayhost = [mail.isp.example]
> # Alternative form:
> # relayhost = [mail.isp.example]:submission
> smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
> ```

- The `smtp_sasl_auth_enable` setting enables client-side authentication. We will configure the client's username and password information in the second part of the example.
- The `relayhost` setting forces the Postfix SMTP to send all remote messages to the specified mail server instead of trying to deliver them directly to their destination.
- In the `relayhost` setting, the "`[`" and "`]`" prevent the Postfix SMTP client from looking up MX (mail exchanger) records for the enclosed name.

- The `relayhost` destination may also specify a non-default TCP port. For example, the alternative form `[mail.isp.example]:submission` tells Postfix to connect to TCP network port 587, which is reserved for email client applications.
- The Postfix SMTP client is compatible with SMTP servers that use the non-standard "`AUTH=method...`" syntax in response to the EHLO command; this requires no additional Postfix client configuration.
- The Postfix SMTP client does not support the obsolete "wrappermode" protocol, which uses TCP port `465` on the SMTP server. See TLS_README[31] for a solution that uses the `stunnel` command.
- With the `smtp_sasl_password_maps` parameter, we configure the Postfix SMTP client to send username and password information to the mail gateway server. As discussed in the next section, the Postfix SMTP client supports multiple ISP accounts. For this reason the username and password are stored in a table that contains one username/password combination for each mail gateway server.

```
/etc/postfix/sasl_passwd:
    # destination                  credentials
    [mail.isp.example]             username:password
    # Alternative form:
    # [mail.isp.example]:submission username:password
```

> ***Important***
> *Keep the SASL client password file in `/etc/postfix`, and make the file read+write only for `root` to protect the username/password combinations against other users. The Postfix SMTP client will still be able to read the SASL client passwords. It opens the file as user `root` before it drops privileges, and before entering an optional chroot jail.*

- Use the `postmap` command whenever you change the `/etc/postfix/sasl_passwd` file.
- If you specify the "`[`" and "`]`" in the `relayhost` destination, then you must use the same form in the `smtp_sasl_password_maps` file.
- If you specify a non-default TCP Port (such as "`:submission`" or "`:587`") in the `relayhost` destination, then you must use the same form in the `smtp_sasl_password_maps` file.

Configuring Sender-Dependent SASL authentication

Postfix supports different ISP accounts for different sender addresses (version 2.3 and

later). This can be useful when one person uses the same machine for work and for personal use, or when people with different ISP accounts share the same Postfix server.

To make this possible, Postfix supports per-sender SASL passwords and per-sender relay hosts. In the example below, the Postfix SMTP client will search the SASL password file by sender address before it searches that same file by destination. Likewise, the Postfix trivial-rewrite(8)[32] daemon will search the per-sender relayhost file, and use the default `relayhost` setting only as a final resort.

> */etc/postfix/main.cf[33]:*
>
> ```
> smtp_sender_dependent_authentication = yes
> sender_dependent_relayhost_maps = hash:/etc/postfix/sender_relay
> smtp_sasl_auth_enable = yes
> smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
> relayhost = [mail.isp.example]
> # Alternative form:
> # relayhost = [mail.isp.example]:submission
> ```

> */etc/postfix/sasl_passwd:*
>
> ```
> # Per-sender authentication; see also /etc/postfix/sender_relay.
> user1@example.com              username2:password2
> user2@example.net              username2:password2
> # Login information for the default relayhost.
> [mail.isp.example]             username:password
> # Alternative form:
> # [mail.isp.example]:submission username:password
> ```

> */etc/postfix/sender_relay:*
>
> ```
> # Per-sender provider; see also /etc/postfix/sasl_passwd.
> user1@example.com              [mail.example.com]:submission
> user2@example.net              [mail.example.net]
> ```

- If you are creative, then you can try to combine the two tables into one single MySQL database, and configure different Postfix queries to extract the appropriate information.
- Specify dbm instead of hash if your system uses dbm files instead of db files. To find out what lookup tables Postfix supports, use the command "postconf -m".
- Execute the command "postmap /etc/postfix/sasl_passwd" whenever you change the

sasl_passwd table.
- Execute the command "postmap /etc/postfix/sender_relay" whenever you change the sender_relay table.

Postfix SMTP/LMTP client policy - SASL mechanism *properties*

Just like the Postfix SMTP server, the SMTP client has a policy that determines which SASL mechanisms are acceptable, based on their properties. The next two sections give examples of how these policies are used.

> | *Property* | *Description* |
> | --- | --- |
> | *noanonymous* | *Don't use mechanisms that permit anonymous authentication.* |
> | *noplaintext* | *Don't use mechanisms that transmit unencrypted username and password information.* |
> | *nodictionary* | *Don't use mechanisms that are vulnerable to dictionary attacks.* |
> | *mutual_auth* | *Use only mechanisms that authenticate both the client and the server to each other.* |

Unencrypted SMTP session

The default policy is stricter than that of the Postfix SMTP server - plaintext mechanisms are not allowed (nor is any anonymous mechanism):

> *`/etc/postfix/main.cf`[34]:*
>
> ```
> smtp_sasl_security_options = noplaintext, noanonymous
> ```

This default policy, which allows no plaintext passwords, leads to authentication failures if the remote server only offers plaintext authentication mechanisms (the SMTP server announces "`AUTH PLAIN LOGIN`"). In such cases the SMTP client will log the following error message:

> ```
> SASL authentication failure: No worthy mechs found
> ```

> ***Note***

> *This same error message will also be logged when the `libplain.so` or `liblogin.so` modules are not installed in the `/usr/lib/sasl2` directory.*

The insecure approach is to lower the security standards and permit plaintext authentication mechanisms:

> */etc/postfix/**main.cf**[35]:*
>
> ```
> smtp_sasl_security_options = noanonymous
> ```

The more secure approach is to protect the plaintext username and password with TLS session encryption. To find out if the remote SMTP server supports TLS, connect to the server and see if it announces STARTTLS support as shown in the example. Information sent by the client (that is, you) is shown in **bold** font.

> ```
> % telnet server.example.com 25
> ...
> 220 server.example.com ESMTP Postfix
> EHLO client.example.com
> 250-server.example.com
> 250-PIPELINING
> 250-SIZE 10240000
> 250-STARTTLS
> ...
> ```

Instead of port 25 (smtp), specify port 587 (submission) where appropriate.

Encrypted SMTP session (TLS)

To turn on TLS in the Postfix SMTP client, see TLS_README[36] for configuration details.

The smtp_sasl_tls_security_options parameter controls Postfix SASL mechanism policy during a TLS-encrypted SMTP session. The default is to copy the settings from the unencrypted session:

> */etc/postfix/**main.cf**[37]:*

```
    smtp_sasl_tls_security_options = $smtp_sasl_security_options
```

A more sophisticated policy allows plaintext mechanisms, but only over a TLS-encrypted connection:

```
/etc/postfix/main.cf[38]:
    smtp_sasl_security_options = noanonymous, noplaintext
    smtp_sasl_tls_security_options = noanonymous
```

Postfix SMTP/LMTP client policy - SASL mechanism *names*

Given the SASL security options of the previous section, the Cyrus SASL library will choose the most secure authentication mechanism that both the SMTP client and server implement. Unfortunately, that authentication mechanism may fail because the client or server is not configured to use that mechanism.

To prevent this, the Postfix SMTP client can filter the names of the authentication mechanisms from the remote SMTP server. Used correctly, the filter hides unwanted mechanisms from the Cyrus SASL library, forcing the library to choose from the mechanisms the Postfix SMTP client filter passes through.

The following example filters out everything but the mechanisms `PLAIN` and `LOGIN`:

```
/etc/postfix/main.cf[39]:
    smtp_sasl_mechanism_filter = plain, login
```

> ***Note***
> *If the remote server does not offer any of the mechanisms on the filter list, authentication will fail.*

We close this section with an example that passes every mechanism except for `GSSAPI` and `LOGIN`:

```
/etc/postfix/main.cf[40]:
    smtp_sasl_mechanism_filter = !gssapi, !login, static:all
```

Building Postfix with SASL support

As mentioned elsewhere, Postfix supports two SASL implementations: Cyrus SASL (SMTP client and server) and Dovecot SASL (SMTP server only). Both implementations can be built into Postfix simultaneously.

Building Dovecot SASL support

These instructions assume that you build Postfix from source code as described in the INSTALL[41] document. Some modification may be required if you build Postfix from a vendor-specific source package.

Support for the Dovecot version 1 SASL protocol is available in Postfix 2.3 and later. At the time of writing, only server-side SASL support is available, so you can't use it to authenticate the Postfix SMTP client to your network provider's server.

Dovecot uses its own daemon process for authentication. This keeps the Postfix build process simple, because there is no need to link extra libraries into Postfix.

To generate the necessary Makefiles, execute the following in the Postfix top-level directory:

```
% make tidy # if you have left-over files from a previous build
% make makefiles CCARGS='-DUSE_SASL_AUTH \
    -DDEF_SERVER_SASL_TYPE=\"dovecot\"'
```

After this, proceed with "`make`" as described in the INSTALL[42] document.

Note

- The `-DDEF_SERVER_SASL_TYPE=\"dovecot\"` is not necessary; it just makes Postfix configuration a little more convenient because you don't have to specify the SASL plug-in type in the Postfix main.cf[43] file (but this may cause surprises when you switch to a later Postfix version that is built with the default SASL type of `sasl`).
- If you also want support for LDAP or TLS (or for Cyrus SASL), you need to merge their `CCARGS` and `AUXLIBS` options into the above command line; see the LDAP_README[44] and TLS_README[45] for details.

```
% make tidy # if you have left-over files from a previous build
% make makefiles CCARGS='-DUSE_SASL_AUTH \
    -DDEF_SERVER_SASL_TYPE=\"dovecot\" \
    ...CCARGS options for LDAP or TLS etc....' \
   AUXLIBS='...AUXLIBS options for LDAP or TLS etc....'
```

Building Cyrus SASL support

Building the Cyrus SASL library

Postfix works with cyrus-sasl-1.5.x or cyrus-sasl-2.1.x, which are available from ftp://ftp.andrew.cmu.edu/pub/cyrus-mail/[46].

> ***Important***
> *If you install the Cyrus SASL libraries as per the default, you will have to create a symlink `/usr/lib/sasl -> /usr/local/lib/sasl` for version 1.5.x or `/usr/lib/sasl2 -> /usr/local/lib/sasl2` for version 2.1.x.*

Reportedly, Microsoft Outlook (Express) requires the non-standard LOGIN and/or NTLM authentication mechanism. To enable these authentication mechanisms, build the Cyrus SASL libraries with:

```
% ./configure --enable-login --enable-ntlm
```

Building Postfix with Cyrus SASL support

These instructions assume that you build Postfix from source code as described in the INSTALL[47] document. Some modification may be required if you build Postfix from a vendor-specific source package.

The following assumes that the Cyrus SASL include files are in `/usr/local/include`, and that the Cyrus SASL libraries are in `/usr/local/lib`.

On some systems this generates the necessary `Makefile` definitions:

Cyrus SASL version 2.1.x

```
% make tidy # if you have left-over files from a previous build
% make makefiles CCARGS="-DUSE_SASL_AUTH -DUSE_CYRUS_SASL \
    -I/usr/local/include/sasl" AUXLIBS="-L/usr/local/lib -lsasl2"
```

Cyrus SASL version 1.5.x

```
% make tidy # if you have left-over files from a previous build
% make makefiles CCARGS="-DUSE_SASL_AUTH -DUSE_CYRUS_SASL \
    -I/usr/local/include" AUXLIBS="-L/usr/local/lib -lsasl"
```

On Solaris 2.x you need to specify run-time link information, otherwise the ld.so run-time linker will not find the SASL shared library:

Cyrus SASL version 2.1.x

```
% make tidy # remove left-over files from a previous build
% make makefiles CCARGS="-DUSE_SASL_AUTH -DUSE_CYRUS_SASL \
    -I/usr/local/include/sasl" AUXLIBS="-L/usr/local/lib \
    -R/usr/local/lib -lsasl2"
```

Cyrus SASL version 1.5.x

```
% make tidy # if you have left-over files from a previous build
% make makefiles CCARGS="-DUSE_SASL_AUTH -DUSE_CYRUS_SASL \
    -I/usr/local/include" AUXLIBS="-L/usr/local/lib \
    -R/usr/local/lib -lsasl"
```

Using Cyrus SASL version 1.5.x

Postfix supports Cyrus SASL version 1.x, but you shouldn't use it unless you are forced to. The makers of Cyrus SASL write:

> *This library is being deprecated and applications should transition to using the SASLv2 library (source: Project Cyrus: Downloads[48]).*

If you still need to set it up, here's a quick rundown:

Read the regular section on SMTP server configurations for the Cyrus SASL framework. The differences are:

- Cyrus SASL version 1.5.x searches for configuration (`smtpd.conf`) in `/usr/lib/sasl/` only. You must place the configuration in that directory. Some systems may have modified Cyrus SASL and put the files into e.g. `/var/lib/sasl/`.
- Use the `saslpasswd` command instead of `saslpasswd2` to create users in `sasldb`.
- Use the `sasldblistusers` command instead of `sasldblistusers2` to find users in `sasldb`.
- In the `smtpd.conf` file you can't use `mech_list` to limit the range of mechanisms offered. Instead, remove their libraries from `/usr/lib/sasl/` (and remember remove those files again when a system update re-installs new versions).

Credits

- Postfix SASL support was originally implemented by Till Franke of SuSE Rhein/Main AG.
- Wietse trimmed down the code to only the bare necessities.
- Support for Cyrus SASL version 2 was contributed by Jason Hoos.
- Liviu Daia added smtpd_sasl_application_name, separated reject_sender_login_mismatch into reject_authenticated_sender_login_mismatch and reject_unauthenticated_sender_login_mismatch, and revised the docs.
- Wietse made another iteration through the code to add plug-in support for multiple SASL implementations, and for reasons that have been lost, also changed smtpd_sasl_application_name into smtpd_sasl_path.
- The Dovecot SMTP server-only plug-in was originally implemented by Timo Sirainen of Procontrol, Finland.
- Patrick Ben Koetter revised this document for Postfix 2.4 and made much needed updates.
- Patrick Ben Koetter revised this document again for Postfix 2.7 and made much needed updates.

1. http://tools.ietf.org/html/rfc4954
2. http://tools.ietf.org/html/rfc2554
3. http://wiki.dovecot.org/
4. http://www.postfix.org/postconf.5.html
5. http://www.postfix.org/TLS_README.html
6. http://git.cyrusimap.org/cyrus-sasl/tree/saslauthd/LDAP_SASLAUTHD
7. http://www.postfix.org/ldap_table.5.html

8. http://www.postfix.org/ldap_table.5.html
9. http://www.postfix.org/ldap_table.5.html
10. http://www.openldap.org/doc/admin
11. http://www.postfix.org/postconf.5.html
12. http://www.postfix.org/postconf.5.html
13. http://www.postfix.org/postconf.5.html
14. http://www.postfix.org/postconf.5.html
15. http://www.postfix.org/postconf.5.html
16. http://www.postfix.org/postconf.5.html
17. http://www.postfix.org/postconf.5.html
18. http://www.postfix.org/postconf.5.html
19. http://www.postfix.org/postconf.5.html
20. http://www.postfix.org/postconf.5.html
21. http://www.postfix.org/postconf.5.html
22. http://www.postfix.org/postconf.5.html
23. http://www.postfix.org/postconf.5.html
24. http://www.postfix.org/postconf.5.html
25. http://www.postfix.org/postconf.5.html
26. http://www.postfix.org/postconf.5.html
27. http://www.postfix.org/postconf.5.html
28. http://www.cpan.org/
29. http://jetmore.org/john/code/gen-auth
30. http://www.postfix.org/postconf.5.html
31. http://www.postfix.org/TLS_README.html
32. http://www.postfix.org/trivial-rewrite.8.html
33. http://www.postfix.org/postconf.5.html
34. http://www.postfix.org/postconf.5.html
35. http://www.postfix.org/postconf.5.html
36. http://www.postfix.org/TLS_README.html

37. http://www.postfix.org/postconf.5.html
38. http://www.postfix.org/postconf.5.html
39. http://www.postfix.org/postconf.5.html
40. http://www.postfix.org/postconf.5.html
41. http://www.postfix.org/INSTALL.html
42. http://www.postfix.org/INSTALL.html
43. http://www.postfix.org/postconf.5.html
44. http://www.postfix.org/LDAP_README.html
45. http://www.postfix.org/TLS_README.html
46. ftp://ftp.andrew.cmu.edu/pub/cyrus-mail/
47. http://www.postfix.org/INSTALL.html
48. http://cyrusimap.web.cmu.edu/downloads.html