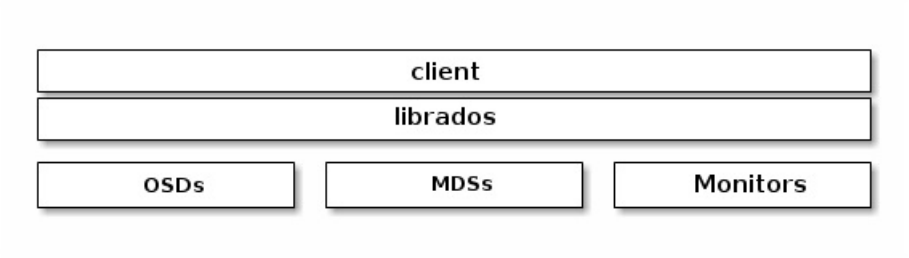
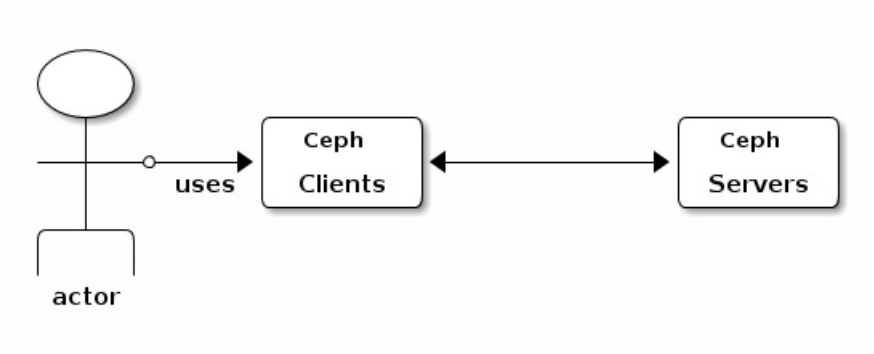
CEPH AUTHENTICATION & AUTHORIZATION

Ceph is a distributed storage system where a typical deployment involves a relatively small quorum of *monitors*, scores of *metadata servers* (MDSs) and many thousands of OSD daemons operating across many hosts/nodes–representing the server portion of the Ceph object store. Ceph clients such as CephFS, Ceph block device and Ceph Gateway interact with the Ceph object store. All Ceph object store clients use the `librados` library to interact with the Ceph object store. The following diagram illustrates an abstract client/server technology stack.



Users are either individuals or system actors such as applications, which use Ceph clients to interact with Ceph server daemons.



For additional information, see our Cephx Guide and ceph-authtool manpage.

CEPH AUTHENTICATION (CEPHX)

Cryptographic authentication has some computational costs, though they should generally be quite low. If the network environment connecting your client and server hosts is very safe and you cannot afford authentication, you can use a Ceph option to turn it off. **This is not generally recommended**, but should you need to do so, details can be found in the Disable Cephx section.

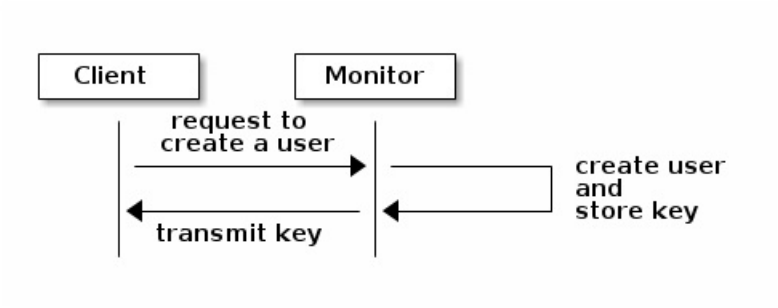
Important: Remember, if you disable authentication, you are at risk of a man-in-the-middle attack altering your client/server messages, which could lead to disastrous security effects.

A key scalability feature of Ceph is to avoid a centralized interface to the Ceph object store, which means that Ceph clients must be able to interact with OSDs directly. To protect data, Ceph provides its cephx authentication system, which authenticates users operating Ceph clients. The cephx protocol operates in a manner with behavior similar to Kerberos.

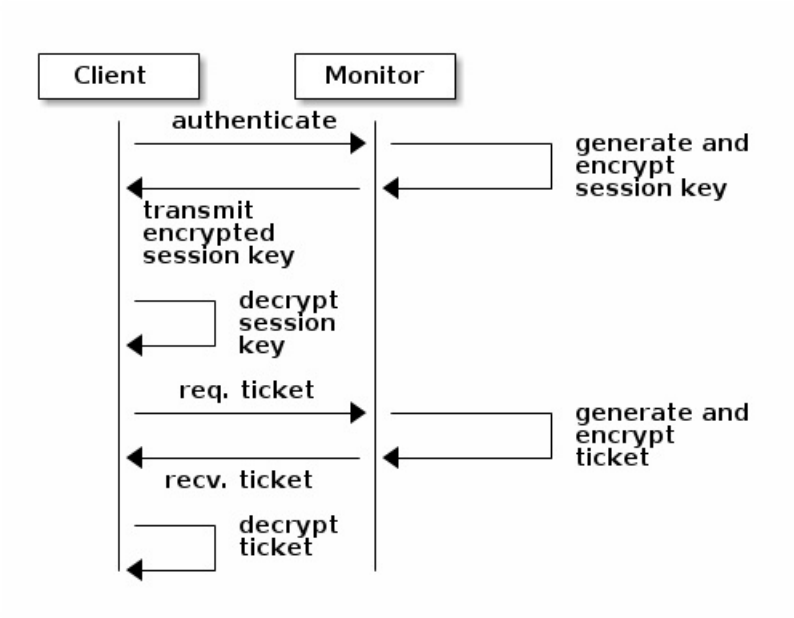
A user/actor invokes a Ceph client to contact a monitor. Unlike Kerberos, each monitor can authenticate users and distribute keys, so there is no single point of failure or bottleneck when using cephx. The monitor returns an authentication data structure similar to a Kerberos ticket that contains a session key for use in obtaining Ceph services. This session key is itself encrypted with the user's permanent secret key, so that only the user can request services from the Ceph monitor(s). The client then uses the session key to request its desired services from the monitor, and the monitor provides the client with a ticket that will authenticate the client to the OSDs that actually handle data. Ceph monitors and OSDs share a secret, so the client can use the ticket provided by the monitor with any OSD or metadata server in the cluster. Like Kerberos, cephx tickets expire, so an attacker cannot use an expired ticket or session key obtained surreptitiously. This form of authentication will prevent attackers with access to the communications medium from either creating bogus messages under another user's identity or altering another user's legitimate messages, as long as the user's secret key is not divulged before it expires.

To use cephx, an administrator must set up users first. In the following diagram, the `client.admin` user invokes `ceph auth get-or-create-key` from the command line to generate a username and secret key. Ceph's `auth` subsystem generates the username and key, stores a copy with the monitor(s) and transmits the user's secret back to the `client.admin` user. This means that the client and the monitor share a secret key.

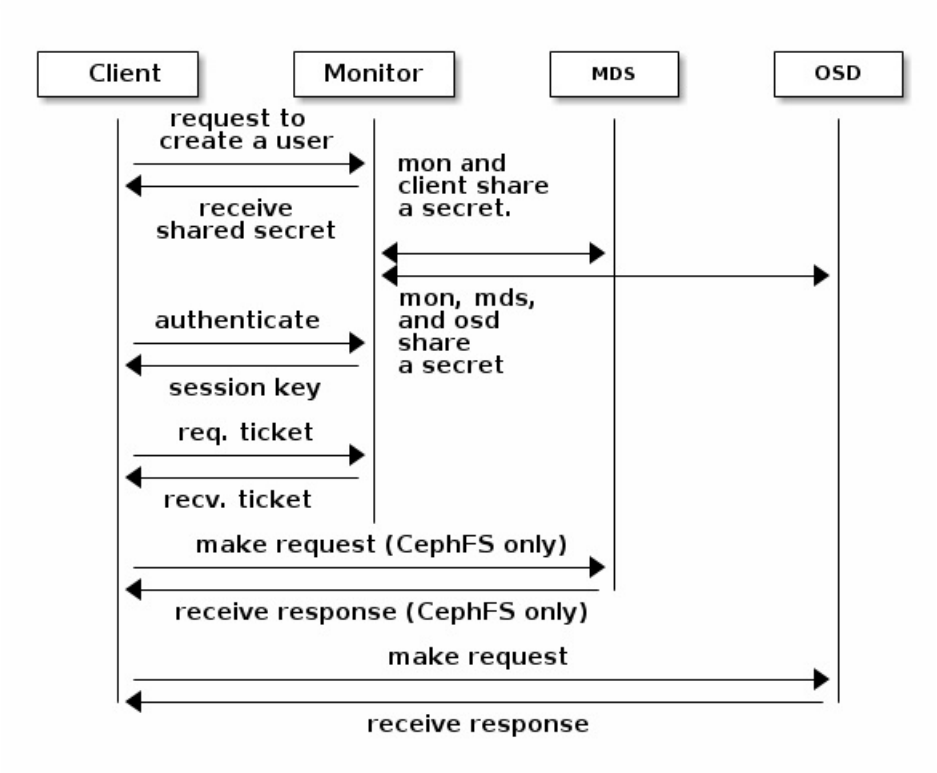
Note: The `client.admin` user must provide the user ID and secret key to the user in a secure manner.



To authenticate with the monitor, the client passes in the user name to the monitor, and the monitor generates a session key and encrypts it with the secret key associated to the user name. Then, the monitor transmits the encrypted ticket back to the client. The client then decrypts the payload with the shared secret key to retrieve the session key. The session key identifies the user for the current session. The client then requests a ticket on behalf of the user signed by the session key. The monitor generates a ticket, encrypts it with the user's secret key and transmits it back to the client. The client decrypts the ticket and uses it to sign requests to OSDs and metadata servers throughout the cluster.



The cephx protocol authenticates ongoing communications between the client machine and the Ceph servers. Each message sent between a client and server, subsequent to the initial authentication, is signed using a ticket that the monitors, OSDs and metadata servers can verify with their shared secret.



The protection offered by this authentication is between the Ceph client and the Ceph server hosts. The authentication is not extended beyond the Ceph client. If the user accesses the Ceph client from a remote host, Ceph authentication is not applied to the connection between the user's host and the client host.

CEPH AUTHORIZATION (CAPS)

Ceph uses the term "capabilities" (caps) to describe authorizing an authenticated user to exercise the functionality of the monitors, OSDs and metadata servers. Capabilities can also restrict access to data within one or more pools.

> **Note:** Ceph uses the capabilities discussed here for setting up and controlling access between various Ceph client and server instances, and are relevant regardless of what type of client accesses the Ceph object store. CephFS uses a different type of capability for files and directories internal to the CephFS filesystem. CephFS filesystem access controls are relevant to CephFS, but not block devices or the RESTful gateway.

A Ceph `client.admin` user sets a user's capabilities when creating the user.

`allow`

Description: Precedes access settings for a daemon. Implies `rw` for MDS only.
Example: `ceph-authtool -n client.foo --cap mds 'allow'`

`r`

Description: Gives the user read access. Required with monitors to retrieve the CRUSH map.
Example: `ceph-authtool -n client.foo --cap mon 'allow r'`

`w`

Description: Gives the user write access to objects.
Example: `ceph-authtool -n client.foo --cap osd 'allow w'`

`x`

Description: Gives the user the capability to call class methods (i.e., both read and write).
Example: `ceph-authtool -n client.foo --cap osd 'allow x'`

`class-read`

Descriptions: Gives the user the capability to call class read methods. Subset of x.
Example: `ceph-authtool -n client.foo --cap osd 'allow class-read'`

`class-write`

Description: Gives the user the capability to call class write methods. Subset of x.
Example: `ceph-authtool -n client.foo --cap osd 'allow class-write'`

*

Description: Gives the user read, write and execute permissions for a particular daemon/pool, and the ability to execute admin commands.
Example: `ceph-authtool -n client.foo --cap osd 'allow *'`

When setting capabilities for a user, Ceph also supports restricting the capabilities to a particular pool. This means you can have full access to some pools, and restricted (or no) access to other pools for the same user. For example:

```
ceph-authtool -n client.foo --cap osd 'allow rwx pool=customer-pool'
```

CEPHX LIMITATIONS

The cephx protocol authenticates Ceph clients and servers to each other. It is not intended to handle authentication of human users or application programs run on their behalf. If that effect is required to handle your access control needs, you must have another mechanism, which is likely to be specific to the front end used to access the Ceph object store. This other mechanism has the role of ensuring that only acceptable users and programs are able to run on the machine that Ceph will permit to access its object store.

The keys used to authenticate Ceph clients and servers are typically stored in a plain text file with appropriate permissions in a trusted host.

Important: Storing keys in plaintext files has security shortcomings, but they are difficult to avoid, given the basic authentication methods Ceph uses in the background. Those setting up Ceph systems should be aware of these shortcomings.

In particular, arbitrary user machines, especially portable machines, should not be configured to interact directly with Ceph, since that mode of use would require the storage of a plaintext authentication key on an insecure machine. Anyone who stole that machine or obtained surreptitious access to it could obtain the key that will allow them to authenticate their own machines to Ceph.

Rather than permitting potentially insecure machines to access a Ceph object store directly, users should be required to sign in to a trusted machine in your environment using a method that provides sufficient security for your purposes. That trusted machine will store the plaintext Ceph keys for the human users. A future version of Ceph may address these particular authentication issues more fully.

At the moment, none of the Ceph authentication protocols provide secrecy for messages in transit. Thus, an eavesdropper on the wire can hear and understand all data sent between clients and servers in Ceph, even if he cannot create or alter them. Further, Ceph does not include options to encrypt user data in the object store. Users can hand-encrypt and store their own data in the Ceph object store, of course, but Ceph provides no features to perform object encryption itself. Those storing sensitive data in Ceph should consider encrypting their data before providing it to the Ceph system.