Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 18 January 2026

G. Lo Presti
CERN
M. B. de Jong
M. Baghbani
Ponder Source
M. Nordin
SUNET

17 July 2025

Open Cloud Mesh
draft-lopresti-open-cloud-mesh-04

Abstract

Open Cloud Mesh is a server federation protocol that is used to notify a Receiving Party that they have been granted access to some Resource. It has similarities with authorization flows such as OAuth, as well as with social internet protocols such as ActivityPub and email.

Open Cloud Mesh only handles the necessary interactions up to the point where the Receiving Party is informed that they were granted access to the Resource. The actual resource access is then left to protocols such as WebDAV and others.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 18 January 2026.

Copyright Notice

Table of Contents

1. Terms

The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document are to be interpreted as described in RFC 2119.

The key words "REQUIRED", "RECOMMENDED", "OPTIONAL" has the same meaning as "MUST", "SHOULD" and "MAY" respectively but are used instead of their counterparts where it makes more grammatical sense (such as marking fields in a JSON object).

We define the following concepts (with some non-normative references to related concepts from OAuth and elsewhere):

* *Resource* - the piece of data or interaction to which access is being granted, e.g. a file, folder, video call, or printer queue

* *Share* - a policy rule stating that certain actors are allowed access to a Resource. Also: a record in a database representing this rule

* *Sending Party* - a person or party who is authorized to create Shares (similar to "Resource Owner" in OAuth)

* *Receiving Party* - a person, group or party who is granted access to the Resource through the Share (similar to "Requesting Party / RqP" in OAuth-UMA)

* *Sending Server* - the server that:

  - holds the Resource ("file server" or "Entreprise File Sync and Share (EFSS) server" role),

  - provides access to it (by exposing at least one "API"),

  - takes the decision to create the Share based on user interface gestures from the Sending Party (the "Authorization Server" role in OAuth)

  - takes the decision about authorizing attempts to access the Resource (the "Resource Server" role in OAuth)

  - sends out Share Creation Notifications when appropriate (see below)

* *Receiving Server* - the server that:

  - receives Share Creation Notifications (see below)

  - actively or passively notifies the receiving user or group of any incoming Share Creation Notification

  - acts as an API client, allowing the receiving user to access the Resource through an API (e.g. WebDAV) of the sending server

* *Sending Gesture* - a user interface interaction from the Sending Party to the Sending Server, conveying the intention to create a Share

* *Share Creation* - the addition of a Share to the database state of the Sending Server, in response to a successful Sending Gesture or for another reason

* *Share Creation Notification* - a server-to-server request from the sending server to the receiving server, notifying the receiving server that a Share has been created

* *FQDN* - Fully Qualified Domain Name, such as "cloud.example.com"

* *OCM Server* - a server that supports OCM.

* *Discovering Server* - a server that tries to obtain information in OCM API discovery

* *Discoverable Server* - a server that tries to supply information in OCM API discovery

* *OCM Address* - a string of the form <Receiving Party's identifier>@<fqdn> which can be used to uniquely identify a user or group "at" an OCM Server. <Receiving Party's identifier> is an opaque string, unique at the server. <fqdn> is the Fully Qualified Domain Name by which the server is identified. This MUST be the domain at which the /.well-known/ocm endpoint of that server is hosted.

* *OCM Notification* - a message from the Receiving Server to the Sending Server or vice versa, using the OCM Notifications endpoint.

* *Invite Message* - out-of-band message used to establish contact between parties and servers in the Invite Flow, containing an Invite Token (see below) and the Invite Sender's OCM Address

* *Invite Sender* - the party sending an Invite

* *Invite Receiver* - the party receiving an Invite

* *Invite Sender OCM Server* - the server holding an address book used by the Invite Sender, to which details of the Invite Receiver are to be added

* *Invite Receiver OCM Server* - the server holding an address book used by the Invite Receiver, to which details of the Invite Sender are to be added

* *Invite Token* - a hard-to-guess string used in the Invite Flow, generated by the Invite Sender OCM Server and linked uniquely to the Invite Sender’s OCM Address

* *Invite Creation Gesture* - gesture from the Invite Sender to the Invite Sender OCM Server, resulting in the creation of an Invite Token.

* *Invite Acceptance Gesture* - gesture from the Invite Receiver to the Invite Receiver OCM Server, supplying the Invite Token as well as the OCM Address of the Invite Sender, effectively allowlisting the Invite Sender OCM Server for sending Share Creation Notifications to the Invite Receiver OCM Server.

* *Invite Acceptance Request* - API call from the Invite Receiver OCM Server to the Invite Sender OCM Server, supplying the Invite Token as well as the OCM Address of the Invite Receiver, effectively allowlisting the Invite Sender OCM Server for sending Share Creation Notifications to the Invite Receiver OCM Server.

* *Invite Acceptance Response* - HTTP response to the Invite Acceptance Request

* *Share Name* - a human-readable string, provided by the Sending Party or the Sending Server, to help the Receiving Party understand which Resource the Share grants access to

* *Share Permissions* - protocol-specific allowances granted to the Receiving Party on the modes of accessing the Resource

* *Share Requirements* - protocol-specific restrictions on the modes of accessing the Resource

* *WAYF Page* - a Where-Are-You-From page is a discovery service used to identify the OCM Server of an Invite Receiver

* *Directory Service* - a third-party service that exposes a list of trusted OCM Servers

2. General Flow

The lifecycle of an Open Cloud Mesh Share starts with prerequisites such as establishing trust, establishing contact, and OCM API discovery.

Then the share creation involves the Sending Party making a Sending Gesture to the Sending Server, the Sending Server carrying out the actual Share Creation, and the Sending Server sending a Share Creation Notification to the Receiving Server.

After this, the Receiving Server MAY notify the Receiving Party and/or the Sending Server, and will act as an API client through which the Receiving Party can access the Resource. After that, the Share may be updated, deleted, and/or reshared.

3. Establishing Contact

Before the Sending Server can send a Share Creation Notification to the Receiving Server, it needs to establish the Receiving Party's OCM Address (containing the Receiving Server's FQDN, and the Receiving Party's identifier), among other things. Some steps may preceed the Sending Gesture, allowing the Sending Party to establish (with some level of trust) the OCM Address of the Receiving Party. In other cases, establishing the OCM Address of the Receiving Party happens as part of the Sending Gesture.

3.1. Direct Entry

The simplest way for this is if the Receiving Party shares their OCM Address with the Sending Party through some out-of-band means, and the Sending Party enters this string into the user interface of the Sending Server, by means of typing or pasting into an HTML form, or clicking a link to a URL that includes the string in some form.

3.2. Address books

The Sending Server MAY offer the Sending Party an address book tool, where OCM Addresses can be stored over time in a labeled and/or searchable way. This decouples the act by which the OCM Address string is passed into the Sending Server's database from the selection of the Receiving Party in preparation for Share Creation.

3.3. Public Link Flow

An interface for anonymously viewing a Resource on the Sending Server MAY allow any internet user to type or paste an OCM address into an HTML form, as a Sending Gesture. This means that the Sending Party and the Receiving Party could be the same person, so contact between them does not need to be explicitly established.

3.4. Public Invite Flow

Similarly, an interface on the Sending Server MAY allow any internet user to type or paste an OCM address into an HTML form, as a Sending Gesture for a given Resource, without itself providing a way to access that particular Resource. A link to this interface could then for instance be shared on a mailing list, allowing all subscribers to effectively request access to the Resource by making a Sending Gesture to the Sending Server with their own OCM Address.

3.5. Invite Flow

Rationale Many methods for establishing contact allow unsolicited contact with the prospective Receiving Party whenever that party’s OCM Address is known. The Invite Flow requires the Receiving Party to explicitly accept it before it can be used, which establishes bidirectional trust between the two parties involved.

OCM Servers MAY enforce a policy to only accept Shares between such trusted contacts, or MAY display a warning to the Receiving Party when a Share Creation Notification from an unknown Sending Party is received

3.5.1. Steps

* the Invite Sender OCM Server generates a unique Invite Token and helps the Invite Sender to create the Invite Message

* the Invite Sender uses some out-of-band communication to send the Invite Message, containing the Invite Token and the Invite Sender OCM Server FQDN, to the Invite Receiver

* the Invite Receiver navigates to the Invite Receiver OCM Server and makes the Invite Acceptance Gesture. This step MAY be facilitated if the Invite Sender OCM Server implements a WAYF Page, such that the Invite Message would include a link to it for the Invite Receiver to navigate to: the Invite Receiver would then be able to indicate their OCM Server and proceed with the Invite Acceptance Gesture without manually copying the Invite Token.

* the Invite Receiver OCM Server discovers the OCM API of the Invite Sender OCM Server using generic OCM API Discovery (see section below)

* the Invite Receiver OCM Server sends the Invite Acceptance Request to the Invite Sender OCM Server

3.5.2. Invite Acceptance Request Details

Whereas the precise syntax of the Invite Message and the Invite Acceptance Gesture will differ between implementations, the Invite Acceptance Request SHOULD be a HTTP POST request:

* to the /invite-accepted path in the Invite Sender OCM Server’s OCM API

* using application/json as the Content-Type HTTP request header

* its request body containing a JSON document representing an object with the following string fields:

  - REQUIRED: recipientProvider - FQDN of the Invite Receiver OCM Server

  - REQUIRED: token - the Invite Token. The Invite Sender OCM Server SHOULD recall which Invite Sender OCM Address this token was linked to

  - REQUIRED: userID - the Invite Receiver’s identifier at their OCM Server

  - REQUIRED: email - non-normative / informational; an email address for the Invite Receiver. Not necessarily at the same FQDN as their OCM Server

  - REQUIRED: name - human-readable name of the Invite Receiver, as a suggestion for display in the Invite Sender’s address book

* using TLS

* using httpsig (https://datatracker.ietf.org/doc/rfc9421)

The Invite Receiver OCM Server SHOULD apply its own policies for trusting the Invite Sender OCM Server before making the Invite Acceptance Request.

Since the Invite Flow does not require either Party to type or remember the userID, this string does not need to be human-memorable. Even if the Invite Receiver has a memorable username at the Invite Receiver OCM Server, this userID that forms part of their OCM Address does not need to match it.

Also, a different userID could be given out to each contact, to avoid correlation of identities.

If the Invite Sender OCM Server implements a WAYF Page, such a page MAY include a fixed list of servers, in addition to, or instead of, a free-text input where any OCM Server can be entered. This is especially useful if the Invite Sender is part of a federation of associated OCM Servers. In order to populate the list of associated OCM Servers, the Invite Sender’s server MAY make use of a Directory Service, which is expected to follow the specification detailed in Appendix C.

Implementors that provide a WAYF Page SHOULD make the URL for the API endpoint of such a Directory Service configurable, allowing the OCM Server to be part of a network of associated OCM Servers. The configuration mechanism MAY allow an OCM Server to be part of multiple networks, thus displaying a union of multiple lists in its WAYF Page.

3.5.3. Invite Acceptance Response Details

The Invite Acceptance Response SHOULD be a HTTP response:

* in response to the Invite Acceptance Request
* using application/json as the Content-Type HTTP response header
* its response body containing a JSON document representing an object with the following string fields:
  - REQUIRED: userID - the Invite Sender’s identifier at their OCM Server
  - REQUIRED: email - non-normative / informational; an email address for the Invite Sender. Not necessarily at the same FQDN as their OCM Server
  - REQUIRED: name - human-readable name of the Invite Sender, as a suggestion for display in the Invite Receiver’s address book

A 200 response status means the Invite Acceptance Request was successful. A 400 response status means the Invite Token is invalid or does not exist. A 403 response status means the Invite Receiver OCM Server is not trusted to accept this Invite. A 409 response status means the Invite was already accepted.

The Invite Sender OCM Server SHOULD verify the HTTP Signature on the Invite Acceptance Request and apply its own policies for trusting the Invite Receiver OCM Server before processing the Invite Acceptance Request and sending the Invite Acceptance Response.

As with the userID in the Invite Acceptance Request, the one in the Response also doesn’t need to be human-memorable, doesn’t need to match the Invite Sender’s username at their OCM Server.

3.5.4. Addition into address books

Following these step, both servers MAY display the name of the other party as a trusted or allowlisted contact, and enable selecting them as a Receiving Party. OCM Servers MAY enforce a policy to only accept Share Creation Notifications from such trusted contacts, or MAY display a warning to users when a Share Creation Notification from an unknown party is received.

Both servers MAY also allowlist each other as a server with which at least one of their users wishes to interact.

Note that Invites act symmetrically, so once contact has been established, both the Invite Sender and the Invite Receiver may take on either the Sending Party or the Receiving Party role in subsequent Share Creation events.

Both parties may delete the other party from their address book at any time without notifying them.

3.5.5. Security Advantages

It is important to underscore the value of the Invite in this scenario, as it provides four important security advantages. First of all, if the Receiving Server blocks Share Creation Notifications from Sending Parties who are not in the address book of the Receiving Party, then this protects the Receiving Party from receiving unsolicited Shares. An attacker could still send the Receiving Party an unsolicited Share, but they would first need to convince the Receiving Party through an out-of-band communication channel to accept their invite. In many use cases, the Receiving Party has had other forms of contact with the Sending Party (e.g. in-person or email back-and-forth). The out-of-band Invite Message thus leverages

the filters and context which the Receiving Party may already benefit from in that out-of-band communication. For instance, a careful Receiving Party may choose to only accept Invites that reach them via a private or moderated messaging platform.

Second, when the Receiving Party accepts the Invite, the Receiving Server knows that the Sending Server they are about to interact with is trusted by the Sending Party, which in turn is trusted by the Receiving Party, which in turn is trusted by them. In other words, one of their users is requesting the allowlisting of a server they wish to interact with, in order to interact with a party they know out-of-band. This gives the Receiving Server reason to put more trust in the Sending Server than it would put into an arbitrary internet-hosted server.

Third, equivalently, the Sending Server knows it is essentially registering the Receiving Server as an API client at the request of the Receiving Party, to whom the right to request this has been traceably delegated by the Sending Party, which is one of its registered users.

Fourth, related to the second one, it removes the partial 'open relay' problem that exists when the Sending Server is allowed to include any Receiving Server FQDN in the Sending Gesture. Without the use of Invites, a Distributed Denial of Service attack could be organised if many internet users collude to flood a given OCM Server with Share Creation Notifications which will be hard to distinguish from legitimate requests without human interaction. An unsolicited (invalid) Invite Acceptance Request is much easier to filter out than an unsolicited (possibly valid, possibly invalid) Share Creation Notification Request, since the Invite Acceptance Request needs to contain an Invite Token that was previously uniquely generated at the Invite Sender OCM server.

4. OCM API Discovery

Introduction After establishing contact as discussed in the previous section, the Sharing User can send the Share Creation Gesture to the Sending Server, providing the Sending Server with the following information:

* Resource to be shared
* Protocol to be offered for access
* Sending Party's identifier
* Receiving Party's identifier

* Receiving Server FQDN
* OPTIONAL: Share Requirements
* OPTIONAL: Share Name
* OPTIONAL: Share Permissions

The next step is for the Sending Server to additionally discover:

* if the Receiving Server is trusted
* if the Receiving Server supports OCM
* if so, which version and with which optional functionality
* at which URL
* the public key the Receiving Server will use for HTTP Signatures (if any)

The Sending Server MAY first perform denylist and allowlist checks on the FQDN.

If a finite allowlist of Receiving Servers exists on the Sending Server side, then this list may already contain all necessary information.

If the FQDN passes the denylist and/or allowlist checks, but no details about its OCM API are known, the Sending Server can use the following process to try to fetch this information from the Receiving Server.

This process MAY be influenced by a VPN connection and/or IP allowlisting.

When OCM API discovery can occur in preparation of a Share Creation Notification, the Sending Server takes on the 'Discovering Server' role and the Receiving Server plays the role of 'Discoverable Server'.

4.1. Process

At the start of the process, the Discovering Server has either an OCM Address, or just an FQDN from for instance the recipientProvider field of an Invite Acceptance Request.

Step 1: In case it has an OCM Address, it should first extract <fqdn> from it (the part after the last @ sign). Step 2: The Discovering Server SHOULD attempt OCM API discovery a HTTP GET request to https://<fqdn>/.well-known/ocm. Step 3: If that results in a valid HTTP response with a valid JSON response body within reasonable time, go to step 7. Step 4: If not, try a HTTP GET with https://<fqdn>/ocm-provider as the URL instead. Step 5: If that results in a valid HTTP response with a valid JSON response body within reasonable time, go to step 7. Step 6: If not, fail. Step 7: The JSON response body is the data that was discovered.

4.2. Fields

The JSON response body offered by the Discoverable Server SHOULD contain the following information about its OCM API:

* REQUIRED: enabled (boolean) - Whether the OCM service is enabled at this endpoint

* REQUIRED: apiVersion (string) - The OCM API version this endpoint supports. Example: "1.2.1"

* REQUIRED: endPoint (string) - The URI of the OCM API available at this endpoint. Example: "https://my-cloud-storage.org/ocm"

* OPTIONAL: provider (string) - A friendly branding name of this endpoint. Example: "MyCloudStorage"

* REQUIRED: resourceTypes (array) - A list of all resource types this server supports in both the Sending Server role and the Receiving Server role, with their access protocols. Each item in this list should itself be an object containing the following fields:

  - name (string) - A supported resource type (file, folder, calendar, contact, ...). Implementations MUST offer support for at least one resource type, where file is the commonly supported one. Each resource type is identified by its name: the list MUST NOT contain more than one resource type object per given name.

  - shareTypes (array of string) - The supported recipient share types. MUST contain "user" at a minimum, plus optionally "group" and "federation". Example: ["user"]

  - protocols (object) - The supported protocols for accessing shared resources of this type. Implementations that offer file resources MUST support at least webdav, any other combination

of resources and protocols is optional. Example: json { "webdav": "/remote/dav/ocm/", "webapp": "/app/ocm/", "talk": "/apps/spreed/api/" } Fields:

- webdav (string) - The top-level WebDAV path at this endpoint. In order to access a remote shared resource, implementations MAY use this path as a prefix, or as the full path (see sharing examples).

- webapp (string) - The top-level path for web apps at this endpoint. This value is provided for documentation purposes, and it SHOULD NOT be intended as a prefix for share requests.

- datatx (string) - The top-level path to be used for data transfers. This value is provided for documentation purposes, and it SHOULD NOT be intended as a prefix. In addition, implementations are expected to execute the transfer using WebDAV as the wire protocol.

- Any additional protocol supported for this resource type MAY be advertised here, where the value MAY correspond to a top-level URI to be used for that protocol.

* OPTIONAL: capabilities (array of string) - The optional capabilities supported by this OCM Server. As implementations MUST accept Share Creation Notifications to be compliant, it is not necessary to expose that as a capability. Example: ["receive-code", "webdav-uri"]. The array MAY include for instance:

  - "enforce-mfa" - to indicate that this OCM Server can apply a Sending Server’s MFA requirements for a Share on their behalf.

  - "webdav-uri" - to indicate that this OCM Server can append a relative URI to the path listed for WebDAV in the appropriate resourceTypes entry

  - "protocol-object" - to indicate that this OCM Server can receive a Share Creation Notification whose protocol object contains one property per supported protocol instead of containing the standard name and options properties.

  - "invites" - to indicate the server would support acting as an Invite Sender or Invite Receiver OCM Server. This might be useful for suggesting to a user that existing contacts might be upgraded to the more secure (and possibly required) invite flow.

- "receive-code" - to indicate that this OCM Server can receive a code as part of a Share Creation Notification, and exchange it for a bearer token at the Sending Server's /token API endpoint.

- "invite-wayf" - to indicate that this OCM Server exposes a WAYF Page to facilitate the Invite flow.

* OPTIONAL: criteria (array of string) - The criteria for accepting a Share Creation Notification. As all Receiving Servers should require the use of TLS in API calls, it is not necessary to expose that as a criterium. Example: ["http-request-signatures", "code"]. The array MAY include for instance:

  - "http-request-signatures" - to indicate that API requests without http signatures will be rejected.

  - "code" - to indicate that API requests without code will be rejected (i.e. the sharedSecret in the protocol details will be ignored).

  - "denylist" - some servers may be blocked based on their IP address

  - "allowlist" - unknown servers may be blocked based on their IP address

  - "invite" - an invite must have been exchanged between the sender and the receiver before a Share Creation Notification can be sent

* OPTIONAL: publicKey (object) - The signatory used to sign outgoing request to confirm its origin. The signatory is optional, but if present, it MUST contain two string fields, id and publicKeyPem. properties:

  - REQUIRED keyId (string) unique id of the key in URI format. The hostname set the origin of the request and MUST be identical to the current discovery endpoint. Example: https://my-cloud-storage.org/ocm#signature

  - REQUIRED publicKeyPem (string) - PEM-encoded version of the public key. Example: "-----BEGIN PUBLIC KEY-----\nMII...QDD\n -----END PUBLIC KEY-----\n"

* OPTIONAL: inviteAcceptDialog (string) - URL path of a web page where a user can accept an invite, when query parameters "token" and "providerDomain" are provided. Implementations that offer the "invites" capability SHOULD provide this URL as well in order to

enhance the UX of the Invite Flow. If for example "/index.php/apps/sciencemesh/accept" is specified here then a WAYF Page SHOULD redirect the end-user to /index.php/apps/sciencemesh/accept?token=zi5kooKu3ivohr9a&providerDomain=example.com.

5. Share Creation Notification

To create a Share, the Sending Server SHOULD make a HTTP POST request

* to the /shares path in the Receiving Server’s OCM API
* using application/json as the Content-Type HTTP request header
* its request body containing a JSON document representing an object with the fields as described below
* using TLS
* using httpsig (https://datatracker.ietf.org/doc/rfc9421)

5.1. Fields

* REQUIRED shareWith (string) Consumer specific identifier of the user, group or federation the provider wants to share the resource with. This is known in advance. Please note that the consumer service endpoint is known in advance as well, so this is no part of the request body. Example: "51dc30ddc473d43a6011e9ebba6ca770@geant.org"
* REQUIRED name (string) Name of the resource (file or folder). Example: "resource.txt"
* OPTIONAL description (string) Optional description of the resource (file or folder). Example: "This is the Open API Specification file (in YAML format) of the Open Cloud Mesh API."
* REQUIRED providerId (string) Identifier to identify the shared resource at the provider side. This is unique per provider such that if the same resource is shared twice, this providerId will not be repeated. Example: 7c084226-d9a1-11e6-bf26-cec0c932ce01
* REQUIRED owner (string) - Provider specific identifier of the user who owns the resource. Example: "6358b71804dfa8ab069cf05ed1b0ed2a@apiwise.nl"

* REQUIRED sender (string) - Provider specific identifier of the user that wants to share the resource. Please note that the requesting provider is being identified on a higher level, so the former remote property is not part of the request body. Example: "527bd5b5d689e2c32ae974c6229ff785@apiwise.nl"

* OPTIONAL ownerDisplayName (string) Display name of the owner of the resource Example: "Dimitri"

* OPTIONAL senderDisplayName (string) Display name of the user that wants to share the resource Example: "John Doe"

* REQUIRED shareType (string) SHOULD have a value of "user", "group", or "federation", to indicated that the first part of the shareWith OCM Address refers to a Receiving Party who is a single user of the Receiving Server, a group of users at the Receiving Servers, or a group of users that is spread out over various servers, including at least one user at the Receiving Server.

* REQUIRED resourceType (string) Resource type (file, folder, calendar, contact, ...)

* OPTIONAL expiration (integer) The expiration time for the OCM share, in seconds of UTC time since Unix epoch. If omitted, it is assumed that the share does not expire.

* OPTIONAL code (string) A nonce to be exchanged for a (potentially short-lived) bearer token at the Sending Server’s /token endpoint.

* REQUIRED protocol (object) JSON object with specific options for each protocol. The supported protocols are: - webdav, to access the data - webapp, to access remote web applications - datatx, to transfer the data to the remote endpoint

Other custom protocols might be added in the future.

In case a single protocol is offered, there are three ways to specify this object:
Option 1: Set the ‘name‘ field to the name of the protocol, and put the protocol details in a field named ‘options‘.
Option 2: Set the ‘name‘ field to the name of the protocol, and put the protocol details in a field carrying the name of the protocol.
Option 3: Set the ‘name‘ field to ‘multi‘, and put the protocol details in a field carrying the name of the protocol.

Option 1 using the ‘options‘ field is now deprecated. Implementations are encouraged to transition to the new optional properties defined below, such that this field may be removed in a future major version of the spec.

When specifying more than one protocol as different ways to access the Share, the ‘name‘ field needs to be set to ‘multi‘.

If multi is given, one or more protocol endpoints are expected to be defined according to the optional properties specified below. Otherwise, at least webdav is expected to be supported, and its options MAY be given in the opaque options payload for compatibility with v1.0 implementations (see examples). Note though that this format is deprecated. Warning: client implementers should be aware that v1.1 servers MAY support both webdav and multi, but v1.0 servers MAY only support webdav.

* Protocol details for webdav MAY contain:

  - REQUIRED uri (string) An URI to access the remote resource. The URI SHOULD be relative, in which case the prefix exposed by the /.well-known/ocm endpoint MUST be used. Absolute URIs are deprecated.

  - OPTIONAL sharedSecret (string) - required if no code field is given for the Share as a whole (see above). An optional secret to be used to access the resource, such as a bearer token. To prevent leaking it in logs it MUST NOT appear in any URI.

  - OPTIONAL permissions (array of strings) - The permissions granted to the sharee. A subset of: - read allows read-only access including download of a copy. - write allows create, update, and delete rights on the resource. - share allows re-share rights on the resource.

  - OPTIONAL requirements (array of strings) - The requirements that the sharee MUST fulfill to access the resource. A subset of: - mfa-enforced requires the consumer to be MFA-authenticated. This MAY be used if the recipient provider

exposes the enforce-mfa capability. - use-code requires the consumer to exchange the given code via a signed HTTPS request. This MAY be used if the recipient provider exposes the receive-code capability.

* Protocol details for webapp MAY contain:

  - REQUIRED uri (string) An URI to a client-browsable view of the shared resource, such that users may use the web applications available at the site. The URI SHOULD be relative, in which case the prefix exposed by the /.well-known/ocm endpoint MUST be used. Absolute URIs are deprecated.

  - REQUIRED viewMode (string) The permissions granted to the sharee. A subset of: - view allows access to the web app in view-only mode. - read allows read and download access via the web app. - write allows full editing rights via the web app.

  - OPTIONAL sharedSecret (string) An optional secret to be used to access the remote web app, for example in the form of a bearer token.

* Protocol details for datatx MAY contain:

  - REQUIRED srcUri (string) An URI to access the remote resource. The URI SHOULD be relative, in which case the prefix exposed by the /.well-known/ocm endpoint MUST be used. Absolute URIs are deprecated.

  - OPTIONAL sharedSecret (string) An optional secret to be used to access the resource, for example in the form of a bearer token. To prevent leaking it in logs it MUST NOT appear in any URI.

  - OPTIONAL size (integer) The size of the file to be transferred from the sending server.

5.2. Decision to Discard

The Receiving Server MAY discard the notification if any of the following hold true:

* the HTTP Signature is missing but the Sending Server does expose a keypair discoverable from the FQDN part of the sender field in the request body

* the HTTP Signature is missing

* the HTTP Signature is not valid

* no keypair is trusted or discoverable from the FQDN part of the sender field in the request body

* the keypair used to generate the HTTP Signature doesn’t match the one trusted or discoverable from the FQDN part of the sender field in the request body

* the Sending Server is denylisted

* the Sending Server is not allowlisted

* the Sending Party is not trusted by the Receiving Party (e.g. no Invite was exchanged and/or the Sending Party’s OCM Address does not appear in the Receiving Party’s address book)

* the Receiving Server is unable to act as an API client for (any of) the protocol(s) listed for accessing the resource

* an initial check shows that the resource cannot successfully be accessed through (any of) the protocol(s) listed

6. Receiving Party Notification

If the Share Creation Notification is not discarded by the Receiving Server, they MAY notify the Receiving Party passively by adding the Share to some inbox list, and MAY also notify them actively through for instance a push notification or an email message.

They could give the Receiving Party the option to accept or reject the share, or add the share automatically and only send an informational notification that this happened.

7. Share Acceptance Notification

In response to a Share Creation Notification, the Receiving Server MAY discover the OCM API of the Sending Server, starting from the <fqdn> part of the sender field in the Share Creation Notification.

If the OCM API of the Sending Server is successfully discovered, the Receiving Server MAY make a HTTP POST request

* to the /notifications path in the Sending Server’s OCM API

* using application/json as the Content-Type HTTP request header

* its request body containing a JSON document representing an object with the fields as described below

* using TLS

* using httpsig (https://datatracker.ietf.org/doc/rfc9421)

7.1. Fields

* REQUIRED notificationType (string) - in a Share Acceptance Notification it MUST be one of:

  - 'SHARE_ACCEPTED'

  - 'SHARE_DECLINED'

* REQUIRED providerId (string) - copied from the Share Creation Notification for the Share this notification is about

* OPTIONAL resourceType (string) - copied from the Share Creation Notification for the Share this notification is about

* OPTIONAL notification (object) - optional additional parameters, depending on the notification and the resource type

For example, a notification MAY be sent by a recipient to let the provider know that the recipient declined a share. In this case, the provider site MAY mark the share as declined for its user(s). Similarly, it MAY be sent by a provider to let the recipient know that the provider removed a given share, such that the recipient MAY clean it up from its database. A notification MAY also be sent to let a recipient know that the provider removed that recipient from the list of trusted users, along with any related share. The recipient MAY reciprocally remove that provider from the list of trusted users, along with any related share.

7.1.1. Receiving Party Notification

If the Share Creation Notification is not discarded by the Receiving Server, they MAY notify the Receiving Party passively by adding the Share to some inbox list, and MAY also notify them actively through for instance a push notification or an email message.

They could give the Receiving Party the option to accept or reject the Share, or add the Share automatically and only send an informational notification that this happened.

8. Resource Access

To access the Resource, the Receiving Server MAY use multiple ways, depending on the body of the Share Creation Notification. The procedure is as follows: 1. The receiver MUST extract the OCM Server FQDN from the sender field of the received share, and MUST query the Discovery (Section 4) endpoint at that address: the resourceTypes[0].protocols.webdav value is the <sender-ocm-path> to be used in step 3. 2. If code is not empty, the receiver SHOULD make a signed POST request to the /token path inside the Sending Server's OCM API, to exchange the code for a short-lived bearer token, and then use that bearer token to access the Resource. 3. If protocol.name = webdav, the receiver SHOULD inspect the protocol.options property. If it contains a sharedSecret, as in the legacy example (https://cs3org.github.io/OCM-API/docs.html?branch=develop&repo=OCM-API&user=cs3org#/paths/~1shares/post), then the receiver SHOULD make a HTTP PROPFIND request to https://<sharedSecret>:@<sender-host><sender-ocm-path>. Note that this access method, based on Basic Auth, is _deprecated_ and may be removed in a future release of the Protocol. 4. Otherwise, if protocol.name = multi, the receiver MUST inspect the protocol.webdav.uri property: if it's a complete URI, the receiver MUST make a HTTP PROPFIND request against it to access the remote resource. If it only contains an identifier <key>, the receiver MUST make a HTTP PROPFIND request to https://<sender-host><sender-ocm-path>/<key> in order to access the remote resource. Additionally, the receiver MUST pass an Authorization: bearer header with either the short-lived bearer token obtained in step 2, if applicable, or the protocol.webdav.sharedSecret value.

In all cases, in case the Shared Resource is a folder and the Receiving Server accesses a resource within that shared folder, it SHOULD append its relative path to that URL. In other words, the Sending Server SHOULD support requests to URLs such as https://<sender-host><sender-ocm-path>/path/to/resource.txt.

Additionally, if protocol.<protocolname>.requirements includes mfa-enforced, the Receiving Server MUST ensure that the Receiving Party has been authenticated with MFA, or prompt the consmer in order to elevate their session, if applicable.

9. Share Deletion

A "SHARE_ACCEPTED" notification followed by a "SHARE_UNSHARED" notification is equivalent to a "SHARE_DECLINED" notification.

Note that the Sending Server MAY at any time revoke access to a Resource (effectively undoing or deleting the Share) without notifying the Receiving Server.

10. Share Updating

Some implementations have experimented with a "RESHARE_CHANGE_PERMISSION"notification, but the payload and side effects such a notification may have are out of scope of this version of this specification. The Receiving Party sending such a notification has no way of knowing if the Sending Party understood and processed the reshare request or not.

11. Resharing

The "REQUEST_RESHARE" and "RESHARE_UNDO" notification types MAY be used by the Receiving Server to persuade the Sending Server to share the same Resource with another Receiving Party. The details of the payload and side effects such a notification may have are out of scope of this version of this specification. Note that the Receiving Party sending such a notification has no way of knowing if the Sending Party understood and processed the reshare request or not.

12. Appendix A: Multi Factor Authentication

If a Receiving Server exposes the capability enforce-mfa, it indicates that it will try and comply with a MFA requirement set on a Share. If the Sending Server trusts the Receiving Server, the Sending Server MAY set the requirement mfa-enforced on a Share, which the Receiving Server MUST honor. A compliant Receiving Server that signals that it is MFA-capable MUST not allow access to a resource protected with the mfa-enforced requirement, if the Receiving Party has not provided a second factor to establish their identity with greater confidence.

Since there is no way to guarantee that the Receiving Server will actually enforce the MFA requirement, it is up to the Sending Server to establish a trust with the Receiving Server such that it is reasonable to assume that the Receiving Server will honor the MFA requirement. This establishment of trust will inevitably be implementation dependent, and can be done for example using a pre approved allow list of trusted Receiving Servers. The procedure of establishing trust is out of scope for this specification: a mechanism similar to the ScienceMesh (https://sciencemesh.io) integration for the Invite (Section 3.5) capability may be envisaged.

13. Appendix B: Request Signing

A request is signed by adding the signature in the headers. The sender also needs to expose the public key used to generate the signature. The receiver can then validate the signature and therefore the origin of the request. To help debugging, it is recommended to also add all properties used in the signature as headers, even if they can easily be re-generated from the payload.

Note: Signed requests prove the identity of the sender but do not encrypt nor affect its payload.

Here is an example of headers needed to sign a request.


```
{
  "(request-target)": "post /path",
  "content-length": 380,
  "date": "Mon, 08 Jul 2024 14:16:20 GMT",
  "digest": "SHA-256=U7gNVUQiixe5BRbp4Tg0xCZMTcSWXXUZI2\\/xtHM40S0=",
  "host": "hostname.of.the.recipient",
  "Signature": "keyId=\"https://author.hostname/key\",algorithm=\"rsa-sha256\",headers=
\"content-length date digest host\",signature=\"DzN12OCS1rsA[...]o0VmxjQooRo6HHabg==\""
}
```


* ’(request-target)’ contains the reached endpoint and the used method,
* ’content-length’ is the total length of the payload of the request,
* ’date’ is the date and time when the request has been sent,
* ’digest’ is a checksum of the payload of the request,
* ’host’ is the hostname of the recipient of the request (remote when signing outgoing request, local on incoming request),
* ’Signature’ contains the signature generated using the private key and details on its generation:
  - ’keyId’ is a unique id, formatted as an url. hostname is used to retrieve the public key via custom discovery
  - ’algorithm’ specify the algorithm used to generate signature
  - ’headers’ specify the properties used when generating the signature

- 'signature' the signature of an array containing the properties listed in 'headers'. Some properties like content-length, date, digest, and host are mandatory to protect against authenticity override.

13.1. How to generate the Signature for outgoing request

After properties are set in the headers, the Signature is generated and added to the list.

This is a pseudo-code example for generating the Signature header for outgoing requests:

```
headers = {
    '(request-target)': 'post /path',
    'content-length': length_of(payload),
    'date': current_gmt_datetime(),  # Use a function to get the current GMT date as 'D,
d M Y H:i:s T'
    'digest': 'SHA-256=' + base64_encode(hash('sha256', utf8_encode(payload))),
    'host': 'recipient-fqdn',
}

signed = ssl_sign(concatenate_with_newlines(headers), private_key, 'sha256')
signature = {
    'keyId': 'sender-fqdn',  # The sending server's FQDN; find its public key through OCM
 API discovery
    'algorithm': 'rsa-sha256',
    'headers': 'content-length date digest host',
    'signature': signed,
}

headers['Signature'] = format_signature(signature)
```

13.2. How to confirm Signature on incoming request

The first step would be to confirm the validity of each properties:

* (request-target) and host are immutable to the type of the request and the local/current host,

* content-length and digest can be re-generated and compared from the payload of the request,

* a maximum TTL must be applied to date and current timestamp,

* regarding data contained in the Signature header:

  - using keyId to get the public key from remote signatory,

- headers is used to generate the clear version of the signature and must contain at least content-length, date, digest and host,

- signature is the encrypted version of the signature.

Here is an example of how to verify the signature using the headers, the signature and the public key:

```
clear = {
    '(request-target)': 'post /path',
    'content-length': length_of(payload),
    'date': 'Mon, 08 Jul 2024 14:16:20 GMT',  # The date used in the verification process
    'digest': 'SHA-256=' + base64_encode(hash('sha256', utf8_encode(payload))),  # Recomp
ute the digest for verification
    'host': 'sender-fqdn',
}

signed = headers['Signature']
verification_result = ssl_verify(concatenate_with_newlines(clear), signed, public_key, 's
ha256')

if not verification_result then
    raise InvalidSignatureException
```

13.3. Validating the payload

Following the validation of the signature, the host should also confirm the validity of the payload, that is ensuring that the actions implied in the payload actually initiated on behalf of the source of the request.

As an example, if the payload is about initiating a new share the file owner has to be an account from the instance at the origin of the request.

14. Appendix C: Directory Service

A third-party Directory Service is a back-end service used to federate multiple OCM Servers and facilitate the Invite flow. It is expected to expose, via anonymous HTTP GET, a JSON document with the following format: * REQUIRED: federation - a human-readable name for the list of OCM Servers exposed by the Directory Service * REQUIRED: servers - a JSON array of objects to describe the list of OCM Servers with the following string fields: * REQUIRED: url - the OCM Server's FQDN * REQUIRED: displayName - a human-readable name for the OCM Server Example: json { "federation" : "The ScienceMesh Directory", "servers" : [{ "url" : "https://ocm-server-1.fqdn", "displayName" : "OCM Server 1" }, { "url" : "https://ocm-server-2.fqdn", "displayName" : "OCM Server 2" }] }

15. Acknowledgements

Our deepest thanks and appreciation go to the people who started the work on what would become this specification in 2015. In particular we want to thank (in alphabetical order) Guido Aben, Russell Albert, Holger Angenent, David Anto, Hrachya Astsatryan, Kurt Bauer, Charles du Jeu, Andreas Eckey, David Gillard, Andranik Hayrapetyan Wahi, Christoph Herzog, David Jericho, Frank Karlitschek, Christian Kracher, Ralph Krimmel, Massimo Lamanna, Simon Leinen, Jari Miettinen, Jakub Moscicki, Frederik Orellana, Vlad Roman, Christian Schmitz, Woojin Seok, Rogier Spoor, Christian Sprajc, Peter Szegedi, Ron Trompert, Benedikt Wegmann and Johnatan Xu.

We would also like to thank Ishank Arora, Gianmaria Del Monte, Jrn Friedrich Dreyer, Hugo Gonzlez Labrador, Maxence Lange, Lovisa Lugnegrd, Sandro Mesterheide, Antoon Prins and Bjrn Schissle for their direct contributions to the specification.

Over the years many more people have been involved in the development of OCM. We would like to thank all of them for their contributions, including Jean-Thomas Acquaviva, Samuel Alfageme Sainz, Karsten Asshauer, Miroslav Bauer, Felix Bhm, Maciej Brzeniak, Diogo Castro, Gavin Charles Kennedy, Jarosaw Czub, Milan Danecek, Michael D’Silva, Lukasz Dutka, Pedro Ferreira, Renato Furter, Klaas Freitag, Raman Ganguly, Eva Gergely, Hilary Goodson, Daniel Halbe, Dave Heyns, Jan Holesovsky, Jan Hornicek, Carina Kemp, Fergus Kerins, Andreas Klotz, Matthias Knoll, Christian Kracher, Mario Lassnig, Claudius Laumanns, Anthony Leroy, Patrick Maier, Vladislav Makarenko, Anna Manou, Rita Meneses, Zheng Meyer-Zhao, Crystal Michelle Chua, Yoann Moulin, Daniel Mller, Frederik Mller, Rasmus Munk, Micha Orzechowski, Jacek Pawel Kitowski, Iosif Peterfi, Alessandro Petraro, Rene Ranger, Angelo Romasanta, David Rousse, Carla Sauvanaud, Klaus Scheibenberger, Christian Schmitz, Marcin Sieprawski, Tilo Steiger, C.D. Tiwari, Alejandro Unger and Tom Wezepoel.

Authors’ Addresses

Giuseppe Lo Presti
CERN
Email: giuseppe.lopresti@cern.ch
URI: http://cern.ch/lopresti

Michiel de Jong
Ponder Source
Email: michiel@pondersource.com
URI: https://pondersource.com

Mahdi Baghbani
Ponder Source
Email: mahdi@pondersource.com
URI: https://pondersource.com

Micke Nordin
SUNET
Email: kano@sunet.se
URI: https://code.smolnet.org/micke