Rucio Token Workflow Evolution

Dimitrios Christidis Martin Barisits

Version	Date	Description
0.1	2023-09-25	Initial public release.

1 Introduction

1.1 State of this Document

This document is a working document and should be interpreted as such. It is expected to evolve together with the development, deployment, and testing of what is described herein. Its primary objective is to explain, to the Rucio community, the collective understanding and planning for token workflows in Rucio. The document should inspire critical scrutiny and the authors appeal to the reader to send them comments, so that it may be improved over time.

1.2 Acknowledgements

This document is the result of many meetings and discussions with a large number of individuals and groups: the Rucio community, the FTS team, the WLCG Authorization working group, the WLCG DOMA Bulk Data Transfers working group, the ATLAS and CMS distributed data management teams, the ESCAPE collaboration, the dCache and XRootD teams, and many others. The authors would like to thank Jaroslav Guenther (CERN) and Rizart Dona (CERN) in particular, who contributed the initial implementation of OAuth/OIDC in Rucio.

2 Central Workflows

2.1 Third-Party-Copy Transfer

The third-party-copy transfer workflow refers to the transfer of one or more files between two or more RSEs. Rucio itself is unable to perform such operations. Instead, it makes use of external services called transfer tools. Though Rucio is designed to be able to support multiple transfer tools, this document will focus exclusively on FTS.

In Rucio, a unit of transfer is called a request. One request encapsulates the transfer of one file to one RSE using one FTS instance. There could be multiple source RSEs

capable of being used for each request. It should be noted that there isn't necessarily a one-to-one relation between a request and its initiator. It could be that two different users simultaneously created two independent replication rules for the same file on the same destination RSE. In such cases, Rucio will schedule only one transfer request and reuse it for all relevant replication rules.

In FTS, the requests from Rucio are modelled into jobs. Each job contains one or more transfers. It is worth classifying the jobs into the following categories:

Single A single transfer from one source RSE to one destination RSE.

Multi-source Multiple transfers of the same logical file from two or more source RSEs to one destination RSE. FTS will use the sources one by one and stop as soon as one transfer is successful.

Multi-hop A transfer from one source RSE to one destination RSE through one or more intermediate RSEs. Each hop is one transfer. FTS will schedule the transfers in sequence until all of them are successful (and stop as soon as one is unsuccessful).

Bulk Multiple unrelated transfers grouped into the same job. FTS will schedule them independently, as if they were part of different single transfer jobs.

In addition, the source RSEs of single and multi-hop jobs may require a staging operation (e.g. tape storage). But when it comes to tokens, neither the different job categories nor the necessity for staging have a drastic effect on the third-party-copy transfer workflow. Hence, the paragraphs that follow will describe the case of a request which results in a single transfer job.

A Rucio instance may be set up to use more than one FTS instance. If so, the choice is deterministic and is based on the configuration of the destination RSE. In order to be able to submit transfer requests to FTS, Rucio must first acquire a token that provides authentication with that particular FTS instance.

A sequence diagram of this first stage is presented in Figure 1. Further description of the individual steps is provided below:

1. Check if any of the tokens in the cache may be used for this FTS instance. If not, make a token request to the identity provider using Rucio's own token ⓡ and store it in the cache.

2. Use this token ⓣ for all operations that require communication with the FTS instance.

The submission of transfer requests is presented in detail in the paragraphs that follow. However, the token may be used for others operations as well (e.g. querying the state of a transfer job or cancelling it altogether).

It should be noted that the process of authenticating to FTS is decoupled from the actual transfer. For example, it should be possible to authenticate with an X.509 certificate but have FTS use tokens for the transfer between the RSEs and vice versa.

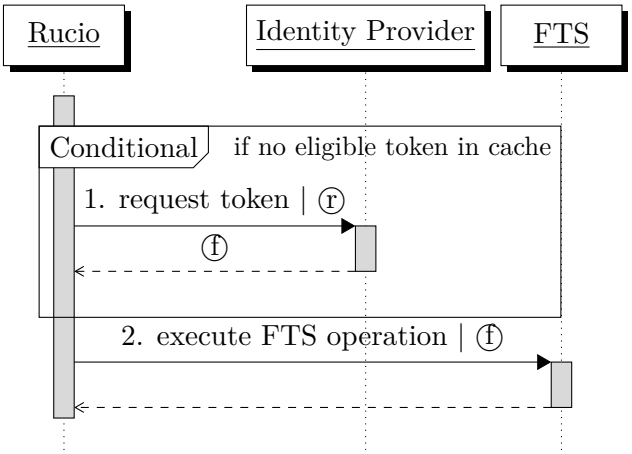


Figure 1: Sequence diagram of the first stage of the third-party-copy transfer workflow: acquiring a token that provides authentication with an FTS instance.

FTS is unable to procure itself the tokens necessary for a transfer. Instead, Rucio must bundle them together with the transfer request. Two tokens are necessary: one to read from the source RSE and one to write to the destination RSE.

A sequence diagram of this second stage is presented in Figure 2. Further description of the individual steps is provided below:

1. Check if any of the tokens stored in the cache may be used for the source RSE. If not, make a token request to the identity provider using Rucio's own token ⓡ and store it in the cache.

2. Check if any of the tokens stored in the cache may be used for the destination RSE. If not, make a token request to the identity provider using Rucio's own token ⓡ and store it in the cache.

Technically, it's in-between this and the following step that Rucio will perform the previously-described authentication stage in order to ensure a valid token ⓕ.

3. Submit a transfer request to FTS. Rucio will use the token ⓕ for authentication purposes and attach the source and destinations tokens ⓢ and ⓓ in the request.

Generally, token lifetimes are expected to be in the order of a few hours. It is impossible to predict how much time will be elapsed from the time a transfer request is submitted to when it is activated. It could vary from a few minutes to many weeks. FTS is responsible for ensuring that the tokens are refreshed so that they can be used when necessary.

A sequence diagram of this third stage is presented in Figure 3. Further description of the individual steps is provided below:

1. Soon after the transfer request is received, FTS obtains a refresh token for the source RSE token ⓢ from the identity provider. For as long as it remains necessary, it periodically checks whether it needs to be refreshed again.

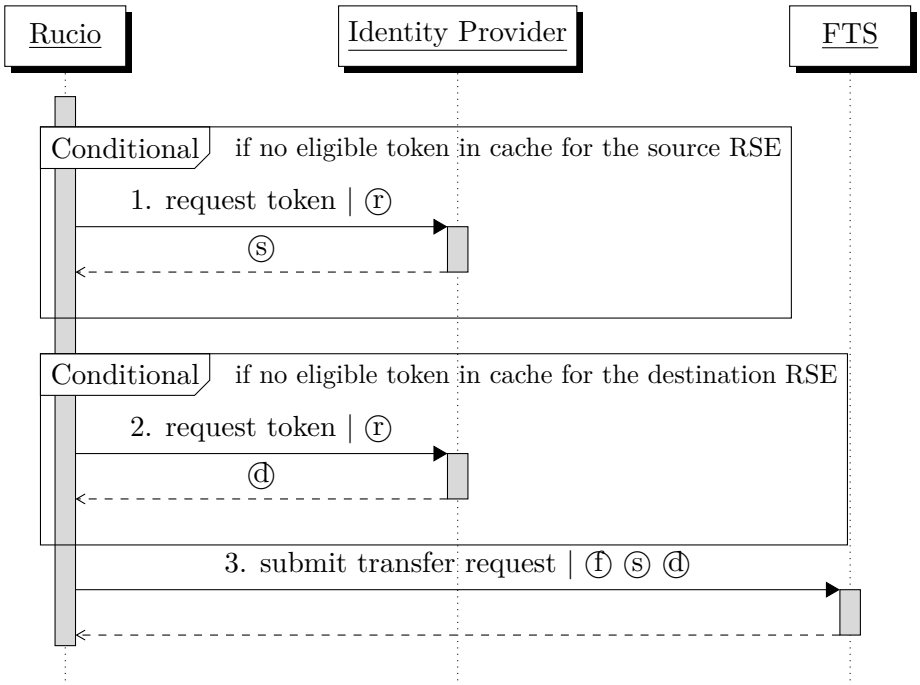


Figure 2: Sequence diagram of the second stage of the third-party-copy transfer workflow: submitting the transfer request to the FTS instance.

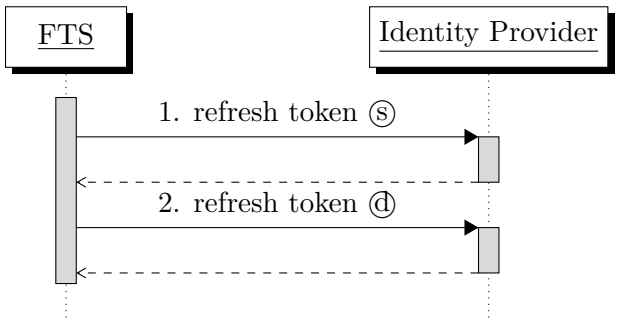


Figure 3: Sequence diagram of the third stage of the third-party-copy transfer workflow: refreshing the provided tokens while there are pending transfers.

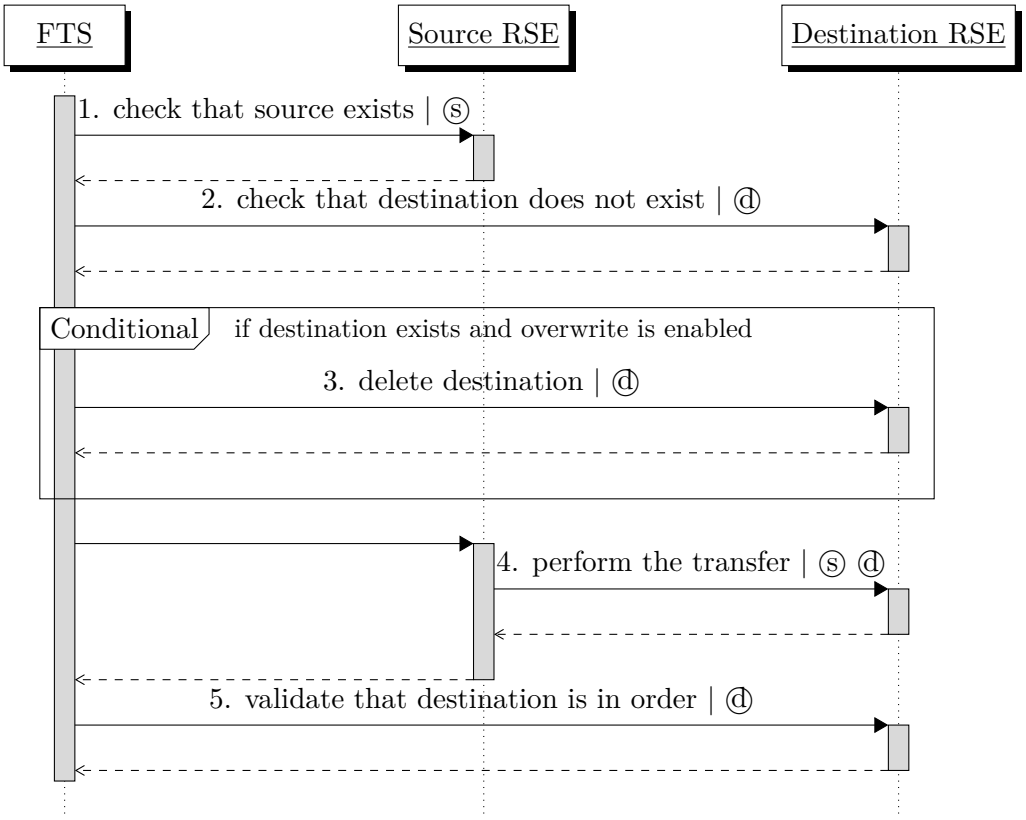


Figure 4: Sequence diagram of the fourth stage of the third-party-copy transfer workflow: performing the actual transfer between the source and the destination RSEs.

2. Likewise for the destination RSE token ⓓ.

Eventually, the transfer job will be activated. Beyond the actual transfer, FTS needs to also perform some additional operations that make use of the same source and destination RSE tokens ⓢ and ⓓ.

A sequence diagram of this fourth stage is presented in Figure 4. Further description of the individual steps is provided below:

1. Check if the file exists at the source RSE using the token ⓢ.

2. Check if the file exists at the destination RSE using the token ⓓ. If it doesn't, then FTS may proceed directly to step 4. If it does but override is disabled for this job, then raise an error and abort.

3. Delete the existing file at the destination RSE using the token ⓓ.

4. Perform the transfer between the source and destination RSEs, using the tokens ⓢ and ⓓ. FTS writes directly to the destination path, not to a temporary file which is to be later renamed. Should the transfer be interrupted, FTS will use the

destination RSE token ⓓ to try to delete the residual file. This is in the interest of avoiding the creation of dark data.

5. Make a final check that the file has been written properly at the destination RSE using the token ⓓ.

On a final but important note, what has been described should be considered an initial implementation. Its main drawback is that the destination RSE token must be capable of deletion. This doesn't offer flexibility for communities who may want to use file-specific tokens for deletion operations done by a third party and RSE-wide tokens for everything else. One of the considerations that had to be taken for this design is that it should be feasible to develop, deploy, and test it before the Data Challenge 2024 exercise. Already there exists a proposal which includes having two tokens for the destination RSE and potentially having FTS request it from Rucio only when necessary (using a callback mechanism). The document will be updated accordingly when the design is finalised and after gaining some experience with the current design.

2.2 Deletion

The deletion workflow refers to the deletion of a replica from an RSE, done centrally by a Rucio daemon. If it's the last replica, then the DID is deleted too. A replica becomes eligible for deletion when it is not locked by any replication rules and it is not used as a source for any active requests. However, the decision to perform the deletion depends on the RSE configuration, whether there are other deletable replicas, and potentially the occupancy levels. The daemon will process RSEs in turn and usually work on a chunk of replicas in each iteration.

A sequence diagram is presented in Figure 5. Further description of the individual steps is provided below:

1. Check if any of the tokens stored in the cache may be used for deletion on this RSE. If not, make a token request to the identity provider using Rucio's own token ⓡ and store it in the cache.

2. Delete each replica in the chunk using the token ⓧ.

3 User Workflows

3.1 Authentication

The authentication workflow refers to the acquisition of a token that identifies the user and can be used to execute Rucio operations. Any changes made to authentication will become perceivable by the user. Consequently, it is important to take into consideration the user experience. By comparison, changes to all other workflows, including download and upload, are expected to be transparent.

For the Rucio Web UI, the standard OAuth workflow is unobtrusive. Instead, this section deals with the Rucio command-line client. Unlike graphical or web applications,

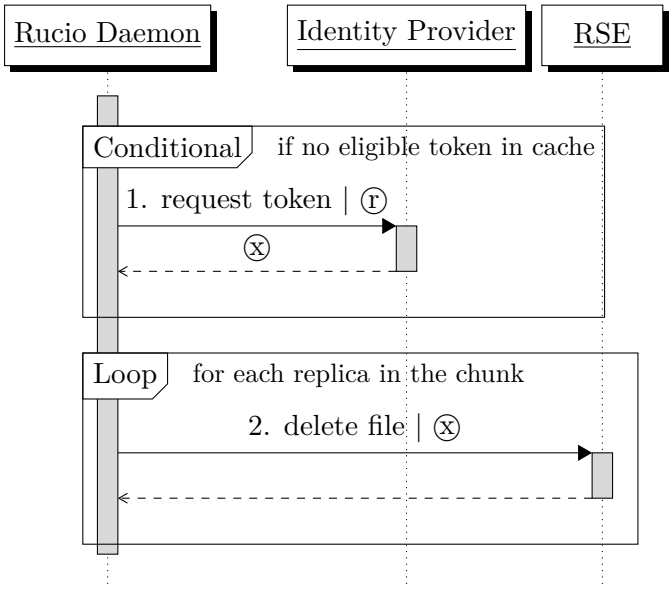


Figure 5: Sequence diagram of the deletion workflow.

the transition from the command-line to a web browser and back can be awkward. If each piece of software implements this workflow internally and the user has to repeat it for each of them, then it will be even more so.

As such, what is envisioned for the future is the use of some external tool that procures tokens in a way that can be used by all other software. This could end up being `oidc-agent`, but for now the placeholder name 'utility X' has been chosen.

A sequence diagram is presented in Figure 6. Further description of the individual steps is provided below:

1. The external utility X makes all necessary steps to acquire a token capable of being used by the Rucio client.

2. The Rucio client then uses this token ⓒ for all operations that require communication with the Rucio server.

It should be highlighted that authentication with OIDC is not a prerequisite for the user workflows that interact with RSEs. One could use an X.509 certificate to authenticate with Rucio and then receive OAuth tokens for operations with the storage. In that case, the client will request and procure a proprietary Rucio token. In the remainder of this document, the Rucio client token ⓒ may refer to either format.

3.2 Download

The download workflow refers to the transfer of one or more files from one or more RSEs to the local machine. The initiator of this workflow can be either a real user or a robot (e.g. a job running on a worker node fetching its input data). By default, the

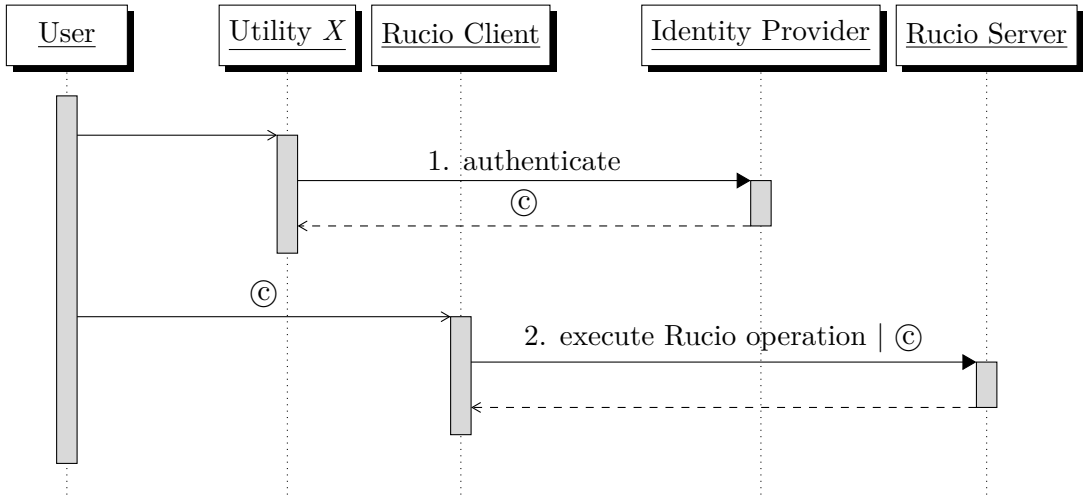


Figure 6: Sequence diagram of the authentication workflow.

Rucio client will request for one or more tokens with the appropriate capabilities from the Rucio server.

A sequence diagram is presented in Figure 7. Further description of the individual steps is provided below:

1. Fetch information on the available replicas of the requested DID using the client token ⓒ.

2. Check if any of the tokens stored in the local client cache may be used for the download. If not, request a new token from the Rucio server using the client token ⓒ and store it in the cache. This will require the introduction of a new endpoint in the REST API. The Rucio server makes a token request to the identity provider using its own token ⓡ. It receives a new token ⓢ and returns it back to the client. It should be noted that the server will also implement some caching mechanism, rendering the request to the identity provider conditional. In addition, the server may refuse to provide a token if the user is not authorised to access the replica (either the DID itself or the RSE on which it is stored).

3. Finally, initiate a transfer of the file from the RSE to the local machine using the token ⓢ. It should be understood that the RSE may differ for each replica.

Alternatively, the initiator may want to bypass the token requests and provide one from the local machine instead. One potential use case for this is workflow-management systems that may wish to handle the tokens internally. It can also prove useful for expert users. The workflow changes only slightly: step 2 is skipped and step 3 uses an externally-provided token.

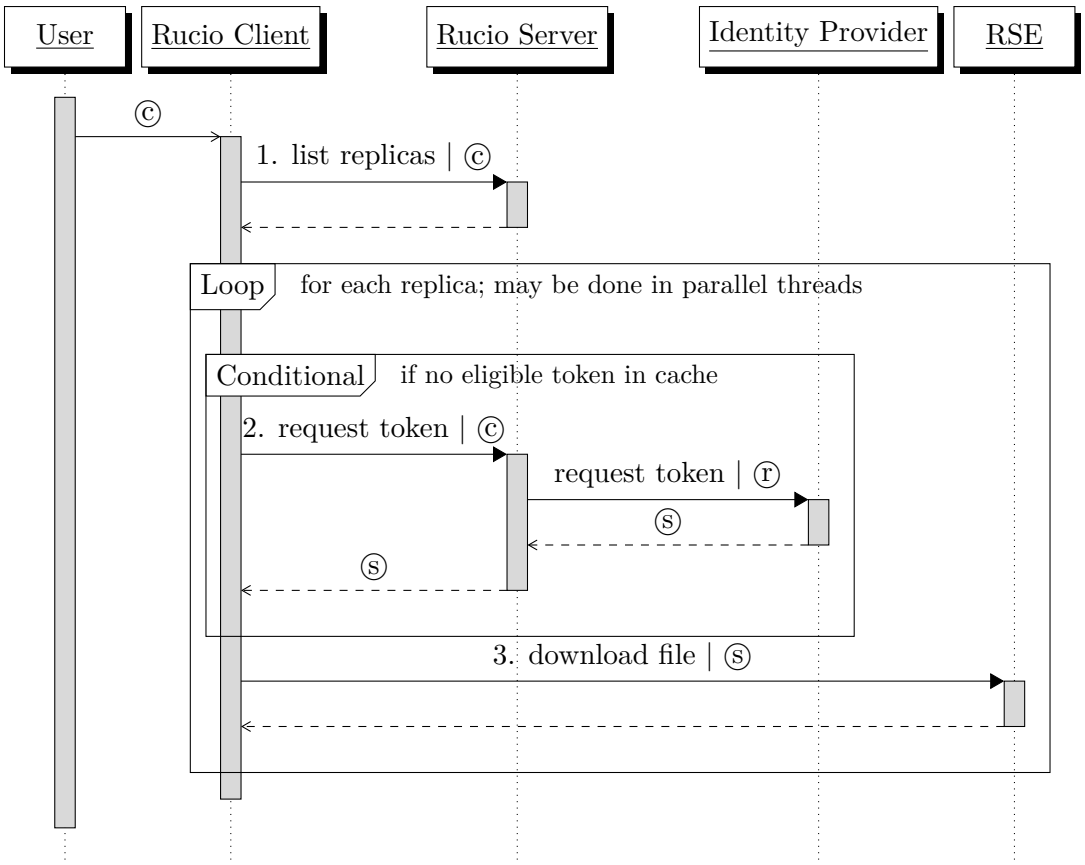


Figure 7: Sequence diagram of the download workflow.

3.3 Upload

The upload workflow refers to the registration and transfer of one or more files from the local machine to an RSE. It is in many ways similar to the download workflow. Once again, the initiator can be either a real user or a robot (e.g. a job running on a worker node pushing its output data). There is some additional complexity rising from the fact that the client might need to delete some files on the RSE prior to transferring the ones from the local machine.

A sequence diagram is presented in Figure 8. Further description of the individual steps is provided below:

1. Make some namespace operations using the token ⓒ. This includes potentially registering a file DID and/or a dataset DID. The new replica is registered in a state that dictates that it is actively being transferred.

2. Check if any of the tokens stored in the local client cache may be used for the upload. If not, request a new token from the Rucio server using the client token ⓒ and store it in the cache. The Rucio server makes a token request to the identity provider using its own token ⓡ. It receives a new token ⓢ and returns it back to the client. Naturally, this will be denied if the user does not have the required privileges. Once again, server-side caching renders the request to the identity provider conditional.

3. Check if the destination path already exists using token ⓢ. In case it does, then it could be a remnant from a previous failed attempt and needs to be deleted. However, the current token is not adequate for such an operation. Otherwise, skip to step 6.

4. Request another token from the Rucio server, one capable of deleting the existing file, using the token ⓒ. This is expected to be restricted to this very specific file. As such, there is no need to implement any caching for it. The Rucio server makes another token request to the identity provider using its own token ⓡ. The privilege check is similar to step 2, but there is an additional check that the replica is not in an available state.

5. Delete the file in the destination path using the newly-received token ⓧ.

6. Initiate a transfer of the file from the local machine to the RSE using the token ⓢ.

7. The Rucio client may implement the upload as a transfer to a temporary location and then a move to the final destination. If so, then rename the file using the token ⓢ. Otherwise, skip to step 8.

8. Finally, if the transfer was successful, then update replica state to reflect so using the token ⓒ.

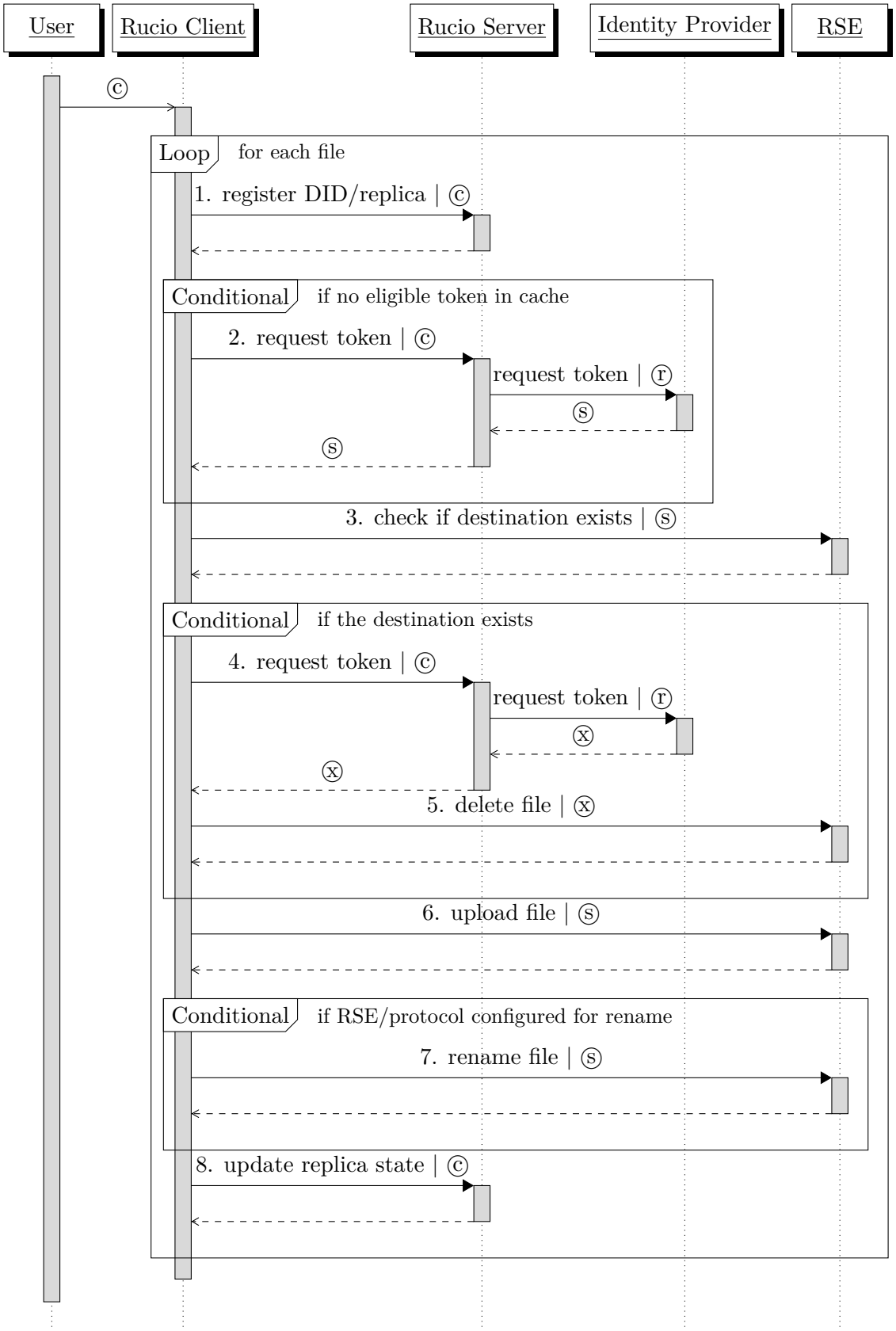


Figure 8: Sequence diagram of the upload workflow.

Similar the download workflow, there is an alternative implementation where the initiator provides a token from the local machine. The changes are as follows: steps 2 and 4 are skipped entirely and steps 3, 5, 6 and 7 use an externally-provided token.

4 Additional Details

4.1 Plugin-Based Approach

It is understood that there cannot be one single solution that satisfies all communities. In the context of tokens, there is a trade-off between security and performance. The stricter the measures in place, the more token requests will be required. By extension, this means higher latency for each operation and higher load on the identity provider.

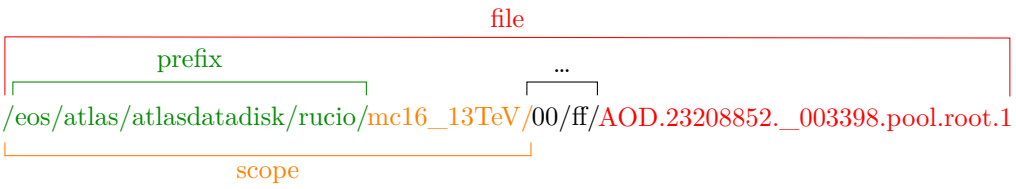
This is something that each community must evaluate and decide on their own. Some decisions can be implemented as simple configuration options. Others, however, will require much greater flexibility. To that end, it is envisioned to construct interfaces in key areas and allow for different implementations as plugins. In a sense, it will resemble what is currently done for permissions. It should be noted that the number of default plugins will be limited. Bespoke implementations will have to be contributed by the communities that need them. The affected areas are described below:

Identity provider This is the external service that handles identity and access management. Initially, only INDIGO IAM will be supported. CILogon should also be expected, but will arrive later.

Token profile Refers to the structure and content of the tokens. For now, only the WLCG token profile is planned to be supported.

Audience restrictions Refers to the intended recipients of the tokens. By default, Rucio will aim to have tokens restricted to a particular storage, using the RSE protocol configuration. However, the WLCG Common JWT Profile allows for a special 'any' value which must be treated as if the token had no audience restriction. This will be configurable per operation.

Scope restrictions Refers to the capabilities of the tokens. This, too, will be configurable per operation. Initially, Rucio will support RSE-level, scope-level, and file-level restrictions.



Namespace access Affects both data and metadata. Initially, Rucio will support access to the entire namespace (as is currently the case).

4.2 Hybrid Functionality

Many communities already have a Rucio instance in production that depends on X.509 certificates. The transition to tokens must be allowed to happen progressively, while maintaining normal operations. There will be some configuration necessary in order for Rucio to be able to procure tokens, but that will not in and of itself enable or force their use.

For workflows related to RSEs (i.e. all except the user authentication), the transition will happen by means of RSE attributes. When set, deletions will happen with tokens. User upload and download too, but that also depends on the version of the Rucio client used.

Third-party-copy transfer will require that all RSEs related to a transfer job have token support enabled. There is a fundamental assumption: that all RSEs support X.509 both at the beginning and during the entirety of the transition. A situation where an RSE supports only tokens but there are other RSEs which support X.509 is not planned to be supported.

On the other hand, user authentication is enabled by means of account identities. Consequently, it is completely independent to the rest of the workflows.