Implementing XRootD/SciToken-Based Access to Lustre Storage at GSI

A First Step Toward Data Federation for FAIR

Rouven Spreckels[1,*], *Anar* Manafov[1,**], *Sören* Fleischer[1,***], *Thorsten* Kollegger[1,****], and *Mohammad* Al-Turany[1,†]

[1]GSI Helmholtzzentrum für Schwerionenforschung GmbH, 64291 Darmstadt, Germany

Abstract. The implementation of a federated access system for GSI's local Lustre storage using XRootD and HTTP(S) protocols will be presented. It aims at ensuring a secure and efficient data access for the diverse scientific communities at GSI. This prototype system is a key step towards integrating GSI/FAIR into a federated data analysis model. We use Keycloak for authentication, which issues SciTokens through OpenID Connect, while LDAP manages local users. After successful login, a JSON Web Token (JWT) is created with appropriate read and write permissions. This token is passed to XRootD's multiuser plugin, which performs the requested operations as the specified user. We also developed an easy-to-use web interface to improve the user experience. This federated access model enhances the security, scalability, and usability of GSI's storage systems, making it a strong solution for modern data management needs.

1 Introduction

The Facility for Antiproton and Ion Research (FAIR) at GSI will generate substantial volumes of scientific data requiring robust management solutions. This paper presents our approach to implementing federated access to GSI's local Lustre [1] storage through XRootD [2] and HTTP(S) [3] protocols, forming a foundation for FAIR's evolving data infrastructure needs.

Our development was guided by three key objectives. First, we aimed to provide flexible data access through both command-line interfaces and web browsers, accommodating the diverse workflows of our scientific community. Second, we sought to implement a secure authentication system by integrating GSI Weblogin with SciTokens, enabling standardized access for employees, guests, and visiting scientists. Third, we leveraged POSIX [4] file ownership and permissions as the underlying access control system for the filesystem to maintain compatibility with existing systems while providing fine-grained security.

This work represents an important step toward integrating GSI/FAIR into broader data federation initiatives. The following sections detail the implementation approach, architectural decisions, and lessons learned during deployment of this system.

*e-mail: r.spreckels@gsi.de
**e-mail: a.manafov@gsi.de
***e-mail: s.fleischer@gsi.de
****e-mail: t.kollegger@gsi.de
†e-mail: m.al-turany@gsi.de

2 Architecture

The system is structured around two public facing hosts shown in figure 1, a generic identity provider (id.gsi.de) powered by Keycloak [5] and a dedicated host (punch2.gsi.de) serving XRootD and the developed Web Frontend and Web Backend. Both hosts retrieve user identities from an internal LDAP service as the single source of truth powered by OpenLDAP [6].

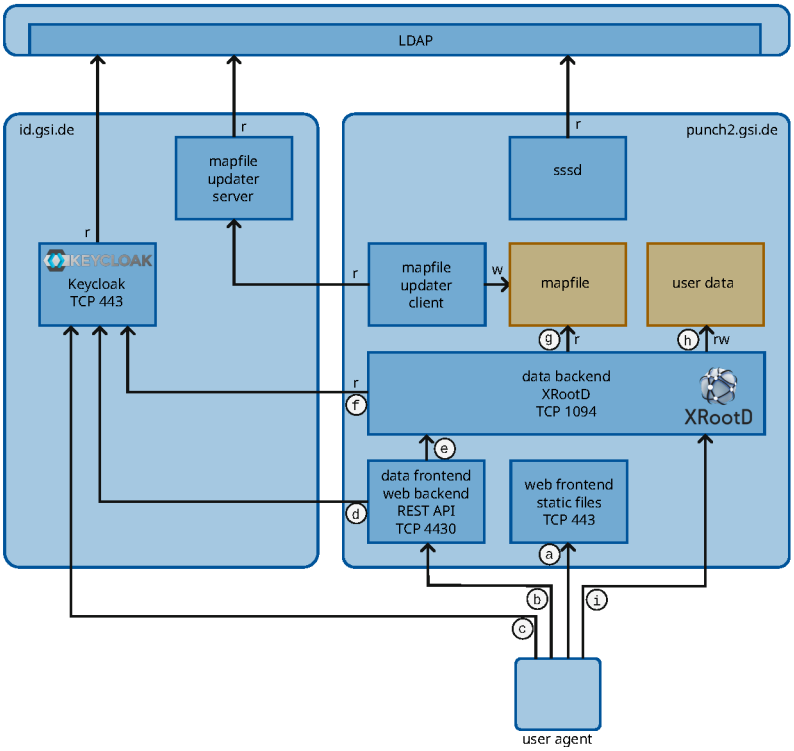

Figure 1. The block diagram shows the involved components and their relations from (a) to (i) as part of the developed workflows. The relations are summarized in table 1. The outermost boxes visualize the two public facing hosts, the internal LDAP server, and the client machine running the user agent (i.e., web browser or CLI terminal).

The identity provider (id.gsi.de) communicates via the OpenID Connect 1.0 (OIDC) standard [7] which is based on the OAuth 2.0 standard [8]. It is a generic service intended to enable single-sign-on/off (SSO) and multi-factor authentication (MFA) for many services at GSI/FAIR. Among others, the OAutch 2.0 standard defines two flows, the Authorization Code Grant which we use for the Web Access Interface and the Resource Owner Password Credentials Grant[1] which we use for the Command-Line Interface. The same host provides an additional internal service mapping the GSI Weblogin accounts to the GSI POSIX accounts. Both identities are stored in the LDAP server but due to different security requirements, they are accessed with different LDAP service accounts and hence a separate mapfile update ser-

[1] Keycloak calls this flow Direct Access Grant. It will be replaced in the future according to RFC 9700 [9].

ⓐ	User's browser downloads client-side JavaScript application from web frontend.
ⓑ	Client-side JavaScript application communicates with web backend.
ⓒ	User's browser displays authentication interface of identity provider as in figure 3.
ⓓ	Web backend communicates with identity provider.
ⓔ	Web backend forwards user requests along with token to XRootD.
ⓕ	XRootD verifies token signature against public key of identity provider.
ⓖ	XRootD looks up POSIX username with Weblogin username claimed in token.
ⓗ	XRootD performs actual file I/O operations on behalf of POSIX username.
ⓘ	User's browser or CURL sends file requests directly to XRootD.

Table 1. Summarizes the component relations from ⓐ to ⓘ as visualized in figure 1.

vice has been deployed. It serves a list[2] with each item comprising the Weblogin as well as POSIX account username.

The dedicated host (punch2.gsi.de) runs multiple services deployed solely for the purpose of the XRootD/SciToken-based access to the Lustre storage at GSI. The mapfile updater client requests a new list every 30 minutes and stores it as semi-static cache for XRootD's multiuser plugin [10]. When the user agent communicates with XRootD, it passes along a JSON Web Token (JWT) [11] adhering to the SciToken scheme [12] as required by the multiuser plugin. This token is issued by the identity provider (id.gsi.de) as described later in the respective user authentication flow. It claims the authenticated Weblogin account's username which is then mapped to the POSIX account's username by the multiuser plugin according to the mapfile. XRootD then performs I/O on behalf of the POSIX account. These accounts are deployed by the SSSD service which has its own LDAP service account privileged to retrieve the POSIX account IDs, usernames, and permissions. The last two remaining services are the newly developed Web Frontend and Web Backend. They implement the Web Access Interface which complements the Command-Line Interface to provide an alternative user workflow.

3 Web Access Interface

The web access interface of our system is structured around two primary components: A frontend web application that users interact with directly, and a backend web service that handles data access requests and authentication. This division allows us to provide a user-friendly interface that leverages modern web technologies while maintaining robust security and performance for data access operations.

3.1 Web Frontend

We implemented the frontend using Vue.js [13], a progressive JavaScript [14] framework for building user interfaces. Vue.js was selected for several key advantages: Its component-based architecture enables reusable and maintainable code; its reactive data binding system

[2]The list is generated on demand when requested by fetching all entries via a dedicated LDAP service account with restricted permissions for retrieving only the accounts' usernames.

simplifies state management; and its lightweight footprint ensures fast load times and responsiveness. These characteristics were essential for creating an intuitive interface that scientists with varying degrees of technical expertise could use effectively without specialized training.

As shown in figure 2, our web interface provides users with a familiar file browser experience while transparently handling the complexity of token-based authentication and XRootD communication. The interface allows users to navigate through directories, view files, and download content with a simple and intuitive design.

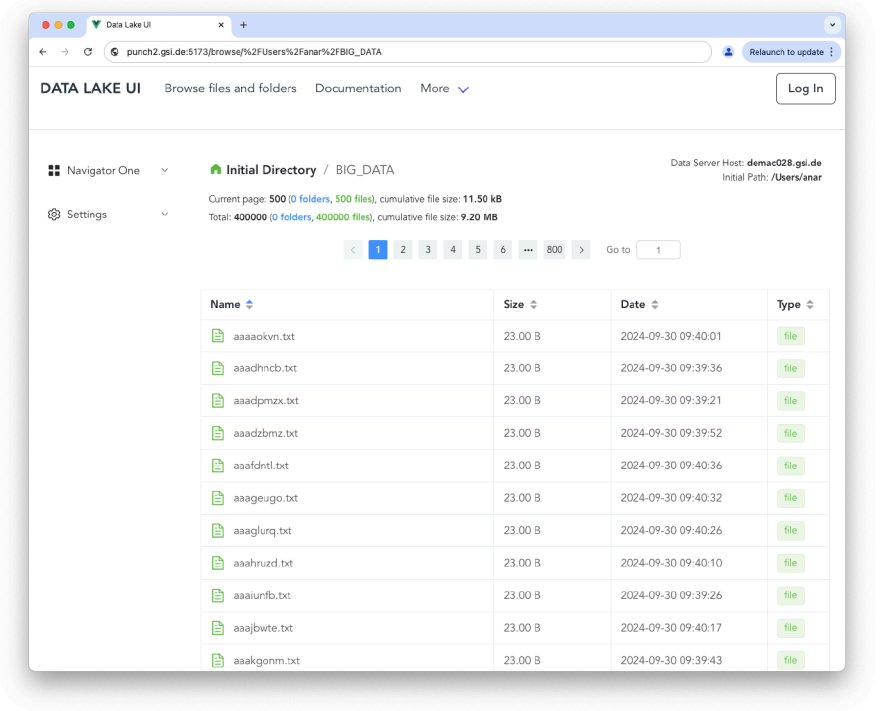

Figure 2. The web interface for browsing files, implemented using Vue.js. The interface provides an intuitive file navigation system that resembles familiar file explorers while adding specific capabilities for data access through XRootD.

3.2 Web Backend

The backend is implemented in Go (Golang) [15], chosen for its exceptional performance in networking applications and concurrent processing. Go's built-in concurrency primitives, such as goroutines and channels, enable efficient handling of multiple simultaneous requests. Additionally, Go's strong typing and compilation to native code help prevent common runtime errors while maintaining excellent performance. These attributes are particularly valuable for data access services where reliability and throughput are critical.

3.3 User Authentication Flow

The user authentication flow involves multiple components working together to provide secure, token-based access. Each step of the following description is visualized in figure 1 and

referred to by a circled letter. ⓐ When a user initially visits the web frontend, JavaScript code is downloaded to their browser. ⓑ This client-side application communicates with the web backend, but since the user does not yet possess an authentication token, they cannot see any data. The interface presents a login option. When users click the login button, they are redirected to ⓒ the GSI Weblogin service where they encounter the authentication interface displayed in figure 3. This Keycloak-powered login screen collects and securely verifies user credentials before issuing the necessary tokens for data access.

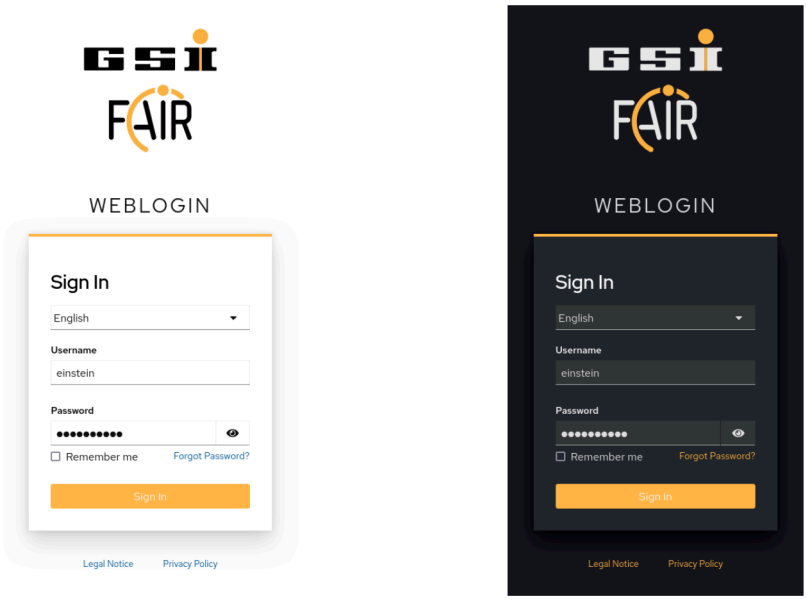

Figure 3. Shows the GSI Weblogin authentication interface with customized light and dark theme of Keycoak's mobile responsive design. Users enter their credentials here to obtain SciTokens for accessing protected data resources.

Upon clicking the login button, the user is redirected to id.gsi.de, GSI's identity provider, where they enter their credentials (i.e., their GSI Weblogin username and password as first factor and either their TOTP[3] or passkey on a FIDO2 [17] security key as second factor). Following successful authentication, id.gsi.de sends an HTTP redirect response back to the user's browser, including an ephemeral one-time code in the GET parameters. ⓑ The client-side code captures this one-time code and forwards it to the web backend. ⓓ The backend then exchanges this code with Keycloak (running on id.gsi.de) for actual authentication tokens, which are returned to the user's browser. ⓑ All subsequent requests from the browser to the REST API include this token, establishing the user's identity and permissions.

3.4 Data Access Flow

The data access flow is facilitated by the web frontend and cached by the web backend whereas the user's browser directly connects to the XRootD server for transferring files. Each step of the following description is visualized in figure 1 and referred to by a circled letter.

[3]The Time-Based One-Time Password (TOTP) algorithm provides short-lived OTP values, which are desirable for enhanced security. [16]

ⓑ When a user requests to browse a directory, ⓔ the web backend forwards this request along with the token to the XRootD server. ⓕ XRootD first downloads the public key from id.gsi.de to verify that the user's token is legitimately signed. ⓖ After verification, XRootD consults its mapfile to determine which POSIX username corresponds to the Weblogin username specified in the token. ⓘ The links to individual files in the directory listing point directly to the XRootD server. When a user clicks on a file, their browser contacts the XRootD server directly. ⓗ The XRootD server, operating with the privileges of the mapped POSIX user, performs the actual file I/O operations on the requested data. Permission checks are handled by the Linux kernel according to standard POSIX rules, ensuring that users can only access files for which their mapped POSIX account has appropriate permissions.

4 Command-Line Interface

The Command-Line interface (CLI) of our system is based on CURL [18]. For the user authentication flow we have developed a SHELL script called TOKIO which requests credentials from the user in order to fetch a SciToken from the identity provider. For the data access flow, CURL is used directly to transfer files with the XRootD server. Each step of the following description is visualized in figure 1 and referred to by a circled letter.

4.1 User Authentication Flow

Initially, the user does not have a token. To obtain a token, the TOKIO script prompts the user to enter their credentials (i.e., their GSI Weblogin username and password as first factor and their TOTP as second factor). ⓒ It sends an HTTP(S) request to id.gsi.de passing along the credentials. If the credentials are accepted, the server responds with a token. This token is shown in figure 4 as part of the prepared "Authorization" HTTP header where the user is the bearer of the token.

Figure 4. Shows the authentication via CLI using the TOKIO script. Only the password prompt is shown as the username is stored in a configuration file and the TOTP is skipped for debugging purpose. This figure reveals no sensitive information as the token has expired and Keycloak's signing keys have been regenerated.

4.2 Data Access Flow

The data access flow utilizes CURL directly. ⓘ The user sends their file request directly to the XRootD server. As shown in figure 5, the user specifies the "Authorization" HTTP header containing the bearer token assigned to the environment variable "$TOKEN".

Figure 5. Shows the file access via CLI using CURL.

ⓕ XRootD downloads the public key of id.gsi.de and verifies that the token is signed by id.gsi.de. ⓖ XRootD reads the mapfile to determine which POSIX username belongs to the Weblogin username claimed in the verified token. ⓗ XRootD performs the actual file I/O on the user data on behalf of the POSIX username specified in the mapfile as shown in figure 6. The permission check is then performed by the Linux kernel according to POSIX rules.

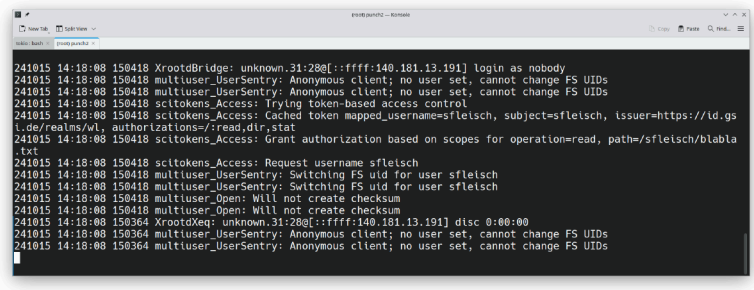

Figure 6. Shows the log of XRootD's multiuser plugin during file access via CLI using CURL.

5 Conclusion

The carefully orchestrated authentication and data access flows maintain security while providing a seamless user experience. By leveraging standard web technologies and established authentication protocols, we've created a system with two complementing interfaces that feel familiar to users while incorporating robust security measures behind the scenes. The Web Access Interface provides a portable file navigation experience familiar to users of a desktop environment's file browser whereas the Command-Line Interface using the same underlying techniques provides expert users a lightweight and flexible interface for automation through user scripts. Designing the federated access model around token-based authentication standards makes GSI's storage systems a strong solution for modern data management needs regarding security, scalability, and usability.

References

[1] OpenSFS and EOFS, The Lustre® file system (2025), https://www.lustre.org
[2] The XRootD Collaboration, XRootD: eXtended ROOT Daemon (2025), https://xrootd.org
[3] M. Thomson, C. Benfield, HTTP/2 (2022). 10.17487/RFC9113
[4] IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7 (2018). 10.1109/IEEESTD.2018.8277153
[5] Keycloak Authors, Open Source Identity and Access Management (2025), https://www.keycloak.org
[6] The OpenLDAP Foundation, Lightweight Directory Access Protocol (2024), https://www.openldap.org
[7] N. Sakimura, J. Bradley, M.B. Jones, B. de Medeiros, C. Mortimore, OpenID Connect Core 1.0 (2014), https://openid.net/specs/openid-connect-core-1_0-final.html
[8] D. Hardt, The OAuth 2.0 Authorization Framework (2012). 10.17487/RFC6749
[9] T. Lodderstedt, J. Bradley, A. Labunets, D. Fett, Best Current Practice for OAuth 2.0 Security (2025). 10.17487/RFC9700
[10] The OSG Consortium, Multi-user OSS and CKS (checksum) plugin for Xrootd (2024), https://github.com/opensciencegrid/xrootd-multiuser
[11] M.B. Jones, J. Bradley, N. Sakimura, JSON Web Token (JWT) (2015). 10.17487/RFC7519
[12] The SciToken Team, Federated Authorization for Distributed Scientific Computing (2023), https://scitokens.org
[13] The Vue Contributors, The Progressive JavaScript Framework (2025), https://vuejs.org
[14] Ecma International, ECMAScript 2024 Language Specification (2024), https://262.ecma-international.org/15.0
[15] The Go Authors, The Go Programming Language (2025), https://go.dev
[16] D. M'Raihi, J. Rydell, M. Pei, S. Machani, TOTP: Time-Based One-Time Password Algorithm (2011). 10.17487/RFC6238
[17] FIDO Alliance, User Authentication Specifications Overview (2025), https://fidoalliance.org/specifications
[18] The CURL Contributors, Command line tool and library for transferring data with URLs (2025), https://curl.se