Computing and Software for Big Science (2025) 9:5
https://doi.org/10.1007/s41781-025-00136-5

BRIEF REPORT

Enabling Secure Shell Access with OpenID Connect

Diana Gudu[1] · **Marcus Hardt**[1] · **Lukas Brocke**[1] · **Gabriel Zachmann**[1]

Received: 15 January 2025 / Accepted: 21 March 2025

Abstract
Secure Shell (SSH) is the de facto standard protocol for accessing remote servers on the command line across a number of use cases, including remote system administration, high-performance computing access, git operations, or system backups via rsync. However, it only supports a limited number of authentication mechanisms, with SSH keys being the most widely used. As federated infrastructures become more prevalent, there is a growing demand for SSH to operate seamlessly and securely in such environments. The use of SSH keys in federated setups poses a number of challenges, since the keys are trusted permanently and can be shared across devices and teams. Mitigations, such as key approval and distribution, make operation at scale complex and error prone. This motivated us to develop a set of tools, collectively referred to as `ssh-oidc`, for facilitating federated identities with SSH by making use of OpenID Connect (OIDC), one of the established protocols used in federated identity management. We support two different approaches: one based on PAM authentication, which works by passing an OIDC access token to the SSH server for authentication, and the other one utilising SSH certificates, which are issued by our online certificate authority in exchange for an access token. Both approaches rely on a central component, `motley_cue`, to handle the mapping of federated identities to Unix accounts on the ssh-server, authorisation, and just-in-time account provisioning. This tool integrates well with user management systems and policies. We also provide client-side tools that automate the process of obtaining and storing the necessary credentials, and ensure a single sign-on experience for the user.

Keywords OIDC · SSH · Authentication · Authorisation · Federated identity · PAM · SSH certificates · Tokens

Introduction

The Secure Shell (SSH) protocol [1] is the most used network protocol for login and execution of commands on remote servers, with OpenSSH [2, 3] being the most prominent and widespread implementation [4]. SSH relies on Unix accounts on the server system.

Federated identities are used to access services or resources from organisations with which users have no direct relationship. They are widely used in research and education communities to provide access to services and resources across organisational boundaries.

Our goal is to enable OpenSSH to directly support authentication with federated identities. For this we use the established OpenID Connect (OIDC) [5] protocol, which is widely used in federated infrastructures. Since the actual identity of the user is already conveyed via OIDC, we want to remove the need to specify usernames in SSH.

Furthermore, we aim to allow administrators to conveniently authorise the correct users, while the burden of user management and authentication remains at their home organisations. Administrators will rather be enabled to specify criteria that federated users need to fulfil to be authorised for using a particular SSH server. This allows reusing criteria, which are already in place for services that support federated identities natively.

✉ Diana Gudu
gudu@kit.edu

Marcus Hardt
hardt@kit.edu

Lukas Brocke
lukas@brocke.net

Gabriel Zachmann
gabriel.zachmann@kit.edu

[1] Karlsruhe Institute of Technology,
Herrmann-von-Helmholtz-Platz 1,
Eggenstein-Leopoldshafen 76344, Germany

Published online: 03 April 2025

Finally, our solution must not require any changes to the OpenSSH source code, since the high security requirements of any SSH implementation require prioritising security over novel features.

Federated Identities

Federated Identity Management (FIM) is a set of technologies and standards that enable users to access resources across organisational boundaries. In a federated environment, users authenticate to their home organisation, which then asserts their identity to the service provider. This process relies on trust relationships between the organisations involved, which are typically established through policies which define the rights and obligations of the parties involved in the federation. Such policies were developed over the last two decades and promoted within eduGAIN [6] and the Authentication and Authorisation for Research and Collaboration (AARC) Policy Development Kit [7].

In the scientific domain, login at the home organisation via eduGAIN is recently being complemented with so-called Community-AAIs [8], such as EGI Check-in [9] or Helmholtz ID [10], that add authorisation information to the set of attributes that describe a user's identity. These attributes will often include information about a user's role or group membership within the Community, membership in Virtual Organisations (VOs), how well an identity was vetted, as well as include information from a user's home organisation.

The use of federated identities is often combined with Single Sign-On (SSO) [11], which allows the user to log-in with a single set of credentials to multiple independent services, often without needing to re-enter their credentials. In OIDC, SSO is achieved through the use of tokens that are trusted across multiple systems, typically implemented as JSON Web Token (JWTs) [12].

Large enterprises, such as Google, Microsoft, or Meta also provide SSO functionality via OIDC or OAuth2, even though they do not follow the federated identity concept. Yet, those identities can also be supported in the solution we present.

SSH Authentication

SSH, as implied by its name, ensures secure connections over insecure networks, by making use of cryptography. It is used to log into a remote machine and execute commands on it. SSH can be used in two ways: interactively, i.e. the user logs in and gets a shell, or non-interactively, i.e. the user logs in and executes a command, the result of which is then returned to the user. The latter is used for example by scp, sftp, rsync, or git.

OpenSSH, the most popular implementation of SSH, supports a number of authentication mechanisms, such as: password authentication [13], a challenge-response method named keyboard-interactive [14], host-based authentication, GSSAPI [15] for Kerberos, as well as public key authentication including SSH certificates [13].

The most widely used method is public key authentication, which is based on asymmetric cryptography. For this method, the user generates a public/private key pair and copies the public key to the server, while keeping the private key locally, optionally protected by a passphrase. After this one-time initial step, the authentication process offers available public keys to the server on every log-in. OpenSSH provides the `ssh-agent` tool to manage the keys and securely load them into memory, without the need to enter a passphrase each time they are used. This tool also supports the forwarding of keys to other machines without storing them on the remote machine.

Nevertheless, public keys have a number of limitations, such as the fact that the keys are not bound to the user's identity. This means that keys can be shared among users, which can be a security risk if the private key is compromised. In addition, the keys are valid indefinitely, unless the administrator explicitly revokes them. This can be a problem if the user leaves the organisation or if the key is compromised. Furthermore, using a passphrase to encrypt the private key cannot be enforced by the server administrator, which means that if the private key file is stolen, the attacker can use it to log into the remote machine without any further authentication. Finally, the keys are based on the Trust-On-First-Use (TOFU) model, which means that the user has to manually verify the fingerprint of the server's public key the first time they connect to it, which can be a security risk if the user is not careful.

SSH certificates address all these limitations. They are based on the same public key cryptography, but they are issued by a Certificate Authority (CA), which ensures that they are bound to a user's identity, optionally with an expiration date, and can be revoked by the CA. In addition, the CA can enforce the use of various security policies. Finally, CAs can also issue certificates for SSH servers, which mitigates the TOFU problem.

The keyboard-interactive method is usually used with Pluggable Authentication Module (PAMs) [16] on the backend, which can be used to implement custom authentication mechanisms, such as two-factor authentication, or to integrate with external authentication systems, such as Lightweight Directory Access Protocol (LDAP) [17]. While it does not mitigate the TOFU problem, it can address the other limitations of public key authentication, if implemented correctly.

Another aspect of SSH authentication is the fact that it is limited to Unix accounts on the server system, which have to exist at the time of the user's login. This makes it difficult to use SSH in a federated environment, since it requires

mapping the user's federated identity to a Unix account on the ssh-server, managing the lifecycle of these accounts, as well as the user's public keys. This can be a cumbersome and error-prone process, especially in large and/or distributed infrastructures, that we aim to address with our solution.

Related Work

There are several other groups in the scientific community that work on the same goal of enabling SSH with federated identities, often with a focus on OIDC. We describe the different approaches and their differences in the following.

A first effort to implement SSH with federated identities was Moonshot [18], which relied on established technologies like RADIUS [19] and SAML [20], which are used by *eduroam* and many national federations. Moonshot used RADIUS to establish a trusted channel via their Trust Router to a SAML based Home Organisation, with ABFAB [21] as the underlying protocol. Furthermore, it used GSS-API for authentication, coupled with a patched version of the OpenSSH server, the latter likely contributing to the slow uptake and subsequent abandonment of the project.

Another similar approach that modified the OpenSSH source code was the GSI-SSH project [22], which relied on a GSS-API mechanism that used X.509 certificates for authentication, implemented by the Globus Toolkit [23]. The patch was never merged by the OpenSSH maintainers, due to prioritising security over new features. Nevertheless, the project was one of the first to translate the identity conveyed in the X.509 user certificate to a Unix user account, so that specifying a username was not required.

This project was later revived under the name of XSEDE OAuth SSH [24, 25], but with a PAM-based approach that uses OAuth tokens from Globus Auth for authentication. User identities are mapped to Unix accounts either by adding a unique Identity Provider (IdP) suffix, or via a mapping file managed by the system administrator. The project also requires a client wrapper around the OpenSSH client.

The SciTokens SSH [26] PAM module builds upon the XSEDE OAuth SSH project, but uses SciTokens [27] instead of opaque OAuth tokens. SciTokens are a token format that is based on JWT, and are used to convey authorisation information. SciTokens SSH is used in the CILogon [28] project, which is a federated identity service for the scientific community.

There are several other PAM-based approaches. What they all have in common is the fact that the username needs to exist on the remote server and be known to the user, since SSH does not allow a PAM module to change the username during the authentication process. These solutions do not address the issue of mapping federated identities to accounts on the ssh server, but rather assume it to be handled outside of their scope.

The solution developed at CESNET [29] uses a PAM module that displays a QR-Code to the user, which redirects the user to a web browser for login via the OIDC device code flow. The solution developed at STFC [30] is based on the CESNET solution, but uses a fork of the original code, developed as part of the IRIS[31] activity. Since the device code flow is not supported by all OpenID Provider (OPs), this limits the usability of the solution. Another limitation is the fact that the user needs to interrupt the SSH login by visiting a web login page on every login.

Another PAM-based solution using the device code flow is the one developed at SURF, called pam-weblogin [32], and it relies on an additional weblogin server. The SURF solution is also interactive, and it works by showing a URL to the user, which they need to visit in a web browser to log in. After a successful login, the user is shown a verification code, which they need to enter in the terminal to complete the login.

An alternative PAM-based implementation by PSNC [33] is prompting the user for an OIDC Access Token (AT), which it verifies with either an OP or another external service. This mitigates the issues of the device code flow, provided that the user can easily obtain an AT. One limitation of this approach is the fact that the OpenSSH client limits the length of the ATs that can be used to 1023 bytes. The solution presented in this paper relies on a patched version of this module, as described in Sect. "pam-ssh-oidc".

A different approach for SSH with OIDC is the so-called "smart shell", where the user shell on the remote machine is replaced with a custom tool that handles the OIDC authentication and authorisation steps. When successful, the smart shell calls the original shell for the user, who is then logged in. While a smart shell can work in both interactive and non-interactive modes, the former is employed by the proprietary solution of AWI[1] and DAASI International.[2]

SSH certificates are used in a solution implemented at DEIC [34]. They use a web browser-based login flow that generates a user-specific command that includes the user token, that the user copies on to the machine from which the SSH client is to be used. The command retrieves the SSH certificate for the authenticated user and makes it available on the client, followed by a non-interactive login. Subsequent logins are non-interactive, as long as the certificate is valid.

Commercial solutions like Teleport [35] or Smallstep SSH [36] also use SSH certificates, albeit without any support for federated identities which are specific to the scientific community. They provide client tools that perform the

[1] https://www.awi.de/.

[2] https://daasi.de/en.

OIDC login and retrieve the SSH certificate, and the SSH certificate is then used for the SSH login.

Requirements

The solution we propose in this paper aims to enable SSH with federated identities, while satisfying a number of requirements. These requirements, collected through discussions with High Performance Computing (HPC) administrators and security experts, are based on the limitations of the current SSH authentication mechanisms, as well as the benefits of federated identities. The solution also aims to be compatible with existing uses of SSH (such as non-interactive sessions, sftp, rsync, git), and does not require any changes to the OpenSSH source code, since this would introduce significant security risks.

Following is a list of requirements that the solution should satisfy:

R1 *Operational security*. The solution should be able to prevent or mitigate common security issues, such as: key sharing among users, key stealing, keys live forever, passphrases cannot be enforced, and the TOFU model.

R2 *Retain benefits of federated identities*:

- *User traceability*. System administrators should be able to trace the actions of users back to their federated identity.
- *Access revocation*. System administrators should be able to revoke access to users based on their federated identity, e.g. if the user leaves the organisation or if their identity is compromised.
- *Interoperability*. The solution should work with existing identity management systems and site policies for user management.
- *SSO*. The solution should allow users to log into multiple systems with a single set of credentials.
- *Authorisation via federated mechanisms*. The solution should allow system administrators to authorise users based on their membership in virtual organisations, or based on other attributes provided by the federated identity provider, such as levels of assurance.

R3 *Support for multiple OPs*. The solution should be able to support multiple OPs for authentication, to allow users from different organisations or even federations to access the same SSH server.

R4 *Just-in-time user-provisioning*. User accounts do not need to exist on the server before the user logs in, and can be created automatically based on the user's federated identity. This implies that the user does not need to know their username on the server, and that the server does not need to know the user's public key.

R5 *Compatibility with existing uses of SSH*. Interactive and non-interactive sessions, scp, sftp, rsync, git.

R6 *No source-code changes*. The solution should not require any changes to the OpenSSH source code, since it would introduce additional challenges to distribute and maintain the solution. For such a security-critical system, users and administrators alike are reluctant, for good reasons, to use a patched version of OpenSSH, whereas currently OpenSSH is widely trusted and provides a high level of security, with regular security updates.

R7 *Minimal changes to user workflows*. The solution should not require users to change their workflows significantly, e.g. by requiring them to use a different SSH client, or by requiring them to perform additional steps during the login process. User experience is often the most overlooked aspect, but crucial for user acceptance, especially in a federated environment, where users are not familiar with the site policies and procedures. The solution should be easy to use at all stages, including the initial setup, the login process, and the management of credentials.

SSH with OIDC

When designing the solution, the following aspects were considered, also depicted in Fig. 1, either by implementing new features and tools, by providing integration points with existing tools, or by providing guidelines for administrators:

A1 *Identity mapping*. Even though the user management on the ssh-server is transparent to the user, SSH still requires an account on the target system. Therefore it is necessary to map federated identities to Unix accounts on the ssh-server system. This is challenging, since Unix usernames are usually limited to 32 ASCII characters, while OIDC identifiers can be much longer and can contain special characters. The mapping must be unique across OPs (a single federated user is mapped

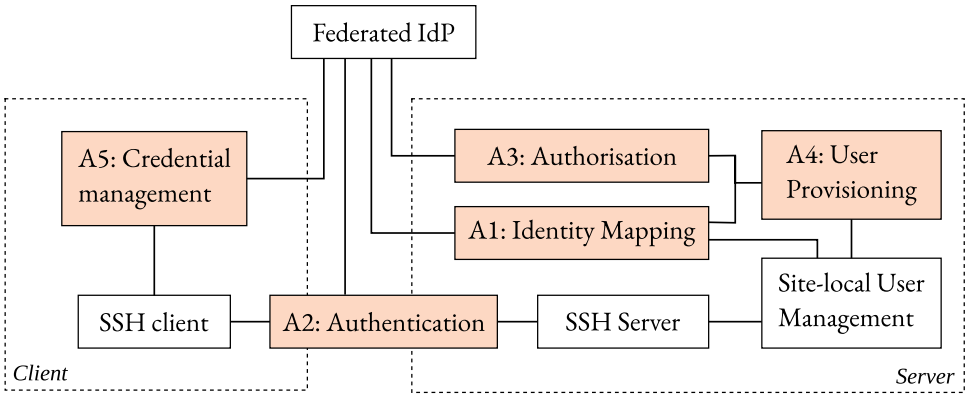

Fig. 1 Considerations for the solution and their interdependencies, including how they fit into a typical SSH login process

to a single Unix account), deterministic (the same user should always be mapped to the same Unix account), and reversible (the Unix account should be able to be mapped back to the federated identity, to ensure traceability).

A2 *Authentication*. Federated users are authenticated at the home organisation using OIDC, but SSH does not support adding new authentication methods without modifying the source code. Therefore, the solution must be able to harmonise the two technologies, either by exchanging OIDC tokens for SSH credentials, or by using SSH mechanisms to transport OIDC tokens to the server for validation.

A3 *Authorisation*. Site administrators must maintain control over who can access the server, but they should be able to base their decisions on federated attributes, such as membership in virtual organisations, or levels of assurance. This requires a mechanism to retrieve these attributes from the federated identity provider, and to map them to the site's authorisation rules.

A4 *User provisioning*. The solution should be able to just-in-time provision (create and update) Unix accounts for federated users, based on their federated identity, including the mapping of federated groups and roles to Unix groups and roles. For the site administrator to maintain control, the solution should provide mechanisms to enforce the site's policies, e.g. for naming and uids, or to integrate with existing user management systems.

A5 *Credential management*. This includes all aspects of obtaining OIDC tokens or SSH credentials, expiry and revocation, securely storing and transporting them.

Our solution, which we call `ssh-oidc`, is composed of a set of different components. Each component addresses one specific task, so that individual components may be combined to support a larger range of use-cases.

One key aspect that differentiates our solution from others is that we use the information contained in the federated identity to identify the corresponding Unix account on the server system. For this we only rely on software components, that we developed ourselves, and in collaboration with external partners.

We support two different approaches to use the solution: one based on the PAM subsystem, and one based on SSH certificates. Both approaches are described in the following sections.

PAM-Based Approach

The first solution relies on a PAM module that is used to authenticate the user based on an OIDC Access Token (AT). The user is prompted for the AT during the SSH login process, and the PAM module verifies it with an OP or another external service. The user does not necessarily need to interrupt the SSH login by visiting a web login page first, provided the user can obtain an AT easily.

Additional tools can be used on client and server side to automate the process, and to provide additional features, such as just-in-time provisioning, as well as to support non-interactive sessions. Their interaction is depicted in Fig. 2.

pam-ssh-oidc

We use a patched version of the PSNC PAM module [33], from hereinafter referred to as `pam-ssh-oidc`. It relies on the challenge-response protocol to implement a conversation function that prompts for an AT. This allows us to prompt the ssh-user for an AT. The patch further enables us to verify the AT with `motley-cue`, one of the tools we developed (described in more detail in Sect. "motley-cue"), instead of the actual OP that issued the token. This allows us to use

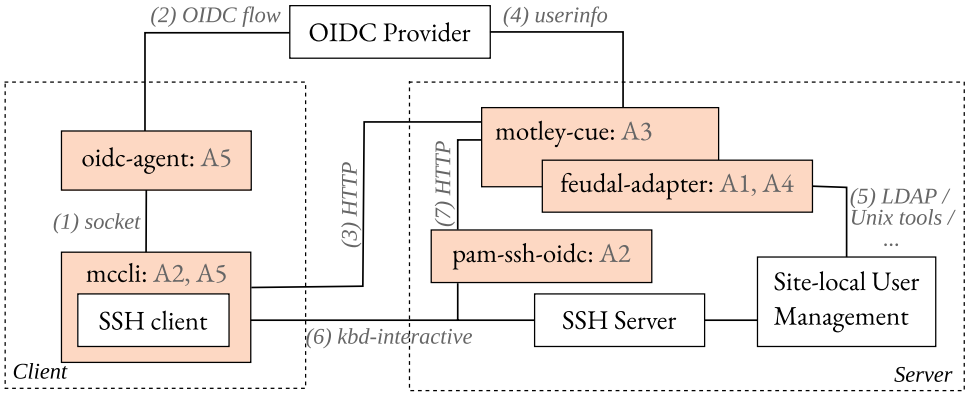

Fig. 2 Architecture of the PAM-based solution. The numbers on the arrows show the steps in a typical SSH login flow when using this solution

any information that `motley-cue` can verify in the PAM prompt. This is important in cases in which the AT is longer than 1023 bytes, a limitation imposed by OpenSSH. In this case, `motley-cue` can issue a one time password (OTP) that is used instead of the AT.

motley-cue

`motley-cue` [37] is a lightweight daemon that runs on the server side, on or nearby the SSH server. Nearby means that `motley-cue` runs in a context that allows it to access the system's user database. Depending on the use-case, read-only or read-write access to the user database may be required.

`motley-cue` provides the following functionalities:

- Ensure authorisation of the user that provided the AT.
- Provide support for multiple OPs, with different authorisation rules.
- Provide/update information to existing mapping between a federated identity and a Unix account.
- Optional: provision an account for the authorised user.
- Verify the AT for a specific Unix user, a functionality aimed to be used by the PAM module.
- Support OTPs in case of access token (AT) longer than the password field.

`motley-cue` exposes its functionality through an OIDC-protected REST interface. A client authenticates with the AT of the user that needs to log in to the SSH server. Details of each functionality are described in the following.

Authorisation

User authorisation is done by allowing the administrator to define access rules based on the user's federated identity, such as membership in virtual organisations or groups, possession of certain entitlements, or levels of assurance. Also, individual users may be authorised. A configuration section per OP allows the administrator to define the authorisation rules for users from that provider.

Authorisation is implemented via the `flaat` library [38], a Python library that provides function decorators for authorising access to REST APIs authenticated with OIDC ATs, with support for multiple API frameworks, such as Flask, FastAPI and AIOHTTP. It also supports multiple OPs, and can be configured to define complex authorisation rules based on OIDC claims.

Account Mapping and Provisioning

`motley-cue` supports native mapping of federated identities to Unix accounts. Native means it uses the information stored directly in a site's local user management system, rather than in an additional database or file managed by `motley-cue`. This allows the site administrator to maintain control over the user accounts and to integrate with existing user management systems, as well as to enforce site policies, such as naming conventions or uids. The mapping is unique across OPs, deterministic, and reversible, to ensure traceability. To uniquely identify the federated user, the `sub` claim is used, together with the OP issuer URL.

`motley-cue` also supports just-in-time provisioning of Unix accounts, based on their federated identity attributes. This includes creating the corresponding Unix account according to the configured naming policies, and adding the user to the correct groups. Generally speaking, the full lifecycle of the mapped Unix accounts can be managed by `motley-cue`, including updating the account, deactivating/reactivating it, or deleting it.

Currently, `motley-cue` supports three backends for user and group management: local Unix accounts (stored in files local to the ssh-server, i.e. `/etc/passwd`), LDAP, and bwIDM (an in-house solution at KIT). In addition, two strategies for naming of user accounts are supported:

- a **friendly** mode, where the username is derived from federated attributes (such as the preferred username—if present—first and last name, or some combination of these attributes) in a user-friendly, readable way, with the constraint that the username is not already assigned to another user.
- a **pooled** mode, where the username is taken from a pool of usernames (defined by a string prefix and an increasing number) and assigned to users in a round-robin fashion, with the same constraint that the username is not already assigned to another user.

Unix groups for each user can also be created based on federated groups and entitlements. This feature is flexible and easily configurable. An administrator can define which groups should be created (all or a subset of the federated groups), and how they should be named. The naming can be configured using regular expressions, which allow for complex transformations of the group names, such as removing prefixes or suffixes, or replacing certain characters, in order to satisfy Unix group naming and length conventions.

Our just-in-time account provisioning also supports a so-called *approval* mode, where the account is created in a pending state, and the site administrator is notified via email to approve or reject the account. This is useful in cases where the site admin wants to review the account before it is activated, or when the account creation requires additional steps.

For the implementation, `motley-cue` relies on `feudalAdapter` [39], a Python library that provides a unified interface to different user management systems, with a plug-in-based architecture that allows for easy extension to new systems.

REST API

`motley-cue` exposes its functionality through an OIDC-protected REST interface, with four main types of endpoints:

- **Info endpoints**: For querying information about the service, such as the supported OPs, privacy policy, or OP-specific settings such as authorisation rules, supported OIDC scopes or required audience. Some of these endpoints can be accessed without authentication, while others require a valid AT from a supported OP.
- **User endpoints**, covering multiple use-cases: i) querying information to which Unix user the bearer of an AT is (going to be) mapped. This includes information such as the username or the provisioning status of the account; ii) triggering the provisioning or the update of the Unix account; and iii) requesting an OTP in case the AT is longer than the allowed length of the password field, 1023 bytes. These endpoints can only be accessed by authorised users.
- **Admin endpoints**: aimed at privileged users at federation-level, such as security officers, who can suspend and unsuspend users, e.g. in case of a security incident. These privileged users need to be authorised individually by the site administrator.
- **System endpoints**: e.g. a user validation endpoint, that can be used by the PAM module during the SSH login process to verify that a given AT belongs to a given Unix user, and that the user is authorised to log in.

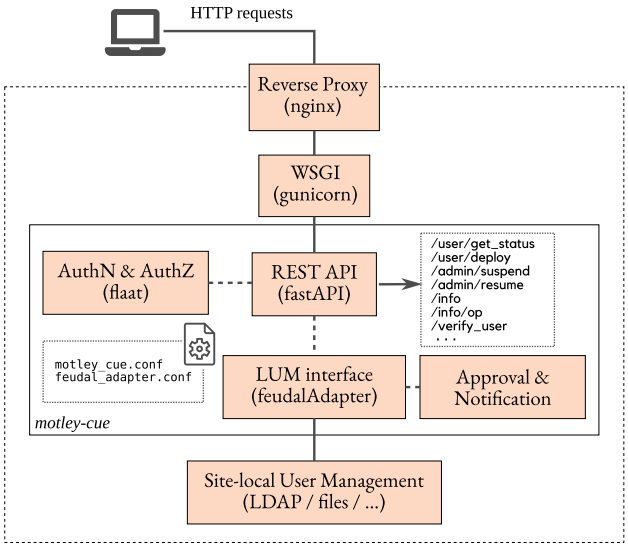

Fig. 3 Architecture and technology stack of the `motley-cue` service. The service is usually deployed behind a reverse proxy that can also handle the SSL encryption, with a Web Service Gateway Interface (WSGI)—a Python-specific middleware allowing Python web applications to communicate with web servers, while also providing features like load-balancing, URL routing, and more. Service settings are configured in two main configuration files: `feudal_adapter.conf` for the user management backend, and `motley_cue.conf` for all other settings, such as authorisation

The API is implemented using FastAPI [40], a modern Python web framework for building APIs based on standard Python type hints. FastAPI provides automatic generation of OpenAPI documentation, as well as interactive API documentation, which makes it easy to use and to integrate with other tools.

Figure 3 shows the architecture and technology stack of the `motley-cue` service.

oidc-agent

The approaches presented in this paper rely on the fact that users need to obtain a valid AT from their OP to authenticate to the SSH server. Any typical OIDC flow redirects the user

to the OP's login page using a web browser, where the user inputs their credentials and consents to the requested scopes. An Identity Token and an AT (and optionally a Refresh Token) are then returned to the user's client. The AT is usually short-lived. This process requires the user to follow an interactive login flow, which is not in line with our goals or our intended usage pattern.

To make the process more convenient, we use `oidc-agent` [41], a command-line tool that allows users to obtain OIDC tokens without interrupting their workflow. `oidc-agent` follows the concept of `ssh-agent`, but instead of SSH key material, it stores an OIDC Refresh Token and the associated `client_id` and `client_secret`, encrypted with a passphrase on the user's client computer. This allows the user to obtain new ATs on the command line without interaction or physical presence at the computer. Having to re-enter their credentials or use a browser, is only required on expiry of a Refresh Token, which ranges between 24 h and one year, depending on the OP.

`oidc-agent` supports multiple account configurations, which correspond to different OP configurations, and hence to different identities of the user. These identities may be referred to with a `shortname`, or using the full issuer URL.

`oidc-agent` also supports agent forwarding over SSH, which allows the remote user to receive tokens from the local agent. This is done by forwarding the Unix domain socket used for communicating with the agent.

mccli

In order to log in to an SSH server with a federated identity, a user needs to go through a number of steps:

1. Obtain an AT from the OP, e.g. using `oidc-agent`.
2. Use the AT to authenticate to the `motley-cue` API and obtain the username that the user is mapped to on the server.
3. If the user is not yet deployed on the server, trigger the provisioning of the user account at the `motley-cue` API using the AT.
4. Optionally, obtain an OTP from the `motley-cue` API if the AT is longer than 1023 bytes.

```
$ mccli <command> <ssh-host> --oidc <oidc-
    agent configuration>
$ mccli ssh ssh-oidc-demo.data.kit.edu --oidc
     egi
```

Fig. 4 Typical use of `mccli` to log in to an SSH server with a federated identity. In this example, `oidc-agent` integration is used to obtain the AT from an already configured account for the EGI OP

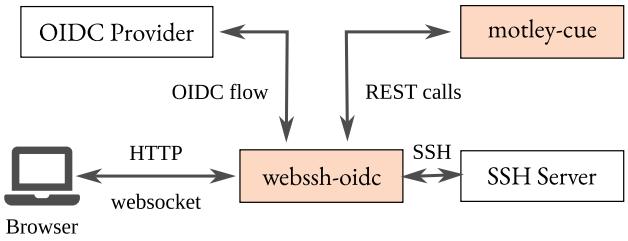

Fig. 5 High-level interactions of `webssh-oidc` with the other relevant components

5. Use the username to log in to the SSH server.
6. When prompted for the AT, place the AT or OTP into the SSH client.

Steps 2 and 3 are necessary on the client-side because on-the-fly user provisioning in the real sense (after authentication, without the user requesting it) is not technically possible.[3] This is due to the so-called *sshd privilege separation* mechanism, where the SSH server saves the initial context at the time of the SSH login, including the SSH username, and ignores any changes during the authentication process, even though PAM supports changing the username. This means the username (or any identifier that NSS can translate to a username) has to be known by the client when initiating the login. Nevertheless, the user can still be provisioned *automatically* and *on-demand* beforehand via `motley-cue`'s REST API with a valid AT, where authorisation is automatically checked.

The "`motley-cue` command-line interface" `mccli` is a client-side tool we developed in order to automate these steps that the user would otherwise need to do manually. Step 6 is implemented using the `pexpect` [42] library to place the AT (or OTP) into the SSH client when prompted, thus enabling a non-interactive SSH login.

`mccli` supports the ssh commands `ssh`, `scp`, and `sftp` and various options for contacting the correct instance of `motley-cue`, as well as for configuring the source of the AT. For details, see [43].

A typical use of `mccli` is shown in Fig. 4. It would—just as with ordinary `ssh`—result in a shell on the remote system.

The drawbacks of the `mccli` approach are that users have to instal two components on their systems: `oidc-agent` and `mccli`. In addition, only selected SSH-based commands are supported. Advanced usage such as piping `stdin/stdout` through SSH is not supported by `mccli`.

[3] At least, not without a custom NSS library, or a smart shell. The former solution was investigated and prototyped, but was not agreed by relevant stakeholders due to the security implications.

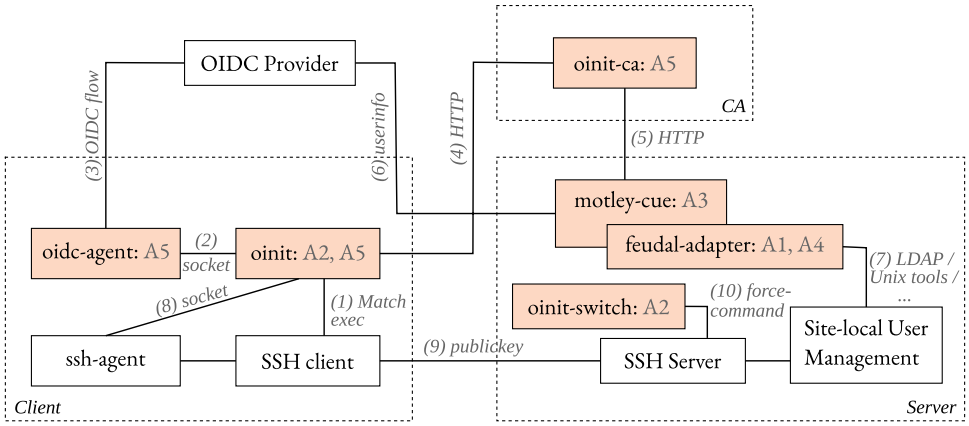

Fig. 6 Architecture of the oinit solution. In this example, the `oinit-ca` runs on a separate host (CA), but multiple configurations are possible. `oinit-ca` can also be run on the same server as `motley-cue`, behind the same reverse proxy, but it is also possible that `oinit-ca`, `motley-cue` and the SSH server are running on different hosts. The depicted configuration with `oinit-ca` and `motley-cue` running on separate hosts has the advantage that a single CA can be used for multiple servers. The numbers on the arrows show the steps in a typical SSH login flow when using this solution

We address this in two ways, as described in Sects. "webssh-oidc" and "SSH Certificate-based Approach".

webssh-oidc

To address the drawback of having to instal additional software on the client-side and reduce the barrier of entry, we also implemented a web-based SSH client that supports log-in to OIDC-enabled SSH servers.

The tool is called `webssh-oidc` and can be run as a web service on or near the server, or on a third-party server that is allowed to access the `motley-cue` API and the SSH servers. Given the configurable nature of the tool, it is possible to run it as a central service that can connect to any infrastructure that allows it.

`webssh-oidc` essentially implements the same steps as `mccli`, but in a web-based environment. The user is redirected to the selected OP's login page, where they log in with their credentials. After the authorisation code flow of OIDC, an AT is made available to the `webssh-oidc` client, which is then used to authenticate to the `motley-cue` API, provision a new account on the ssh-server if necessary, and return the associated username to the client. The client then uses this username to log in to the SSH server via websocket communication, and the AT is used to authenticate the user. The user then has access to a shell on the remote system in their browser.

The entire process is implemented in the client- and server-side components of `webssh-oidc`, and the user only needs a web browser. The burden of installing additional software is thus removed from the user and the effort of running the `webssh-oidc` service will fall on the administrator of the `motley-cue`/SSH server, or can be delegated to a third party. Figure 5 describes the high-level interactions of `webssh-oidc` with the other components.

For the implementation, `webssh-oidc` relies on multiple existing Javascript libraries for OIDC authentication, websocket communication, and web-based terminal emulation. We also provide a demo version of this service for testing.[4]

SSH Certificate-Based Approach

As an alternative, we also provide a solution based on SSH certificates, which addresses some of the limitations of the PAM-based approach. These limitations include security issues (such as the TOFU problem), as well as usability issues (the need to use helper tools on the client-side instead of vanilla SSH client, or the lack of support for advanced SSH command lines or tools that rely on SSH, such as rsync).

The solution, which relies on SSH certificates as an SSH authentication mechanism, consists of a suite of tools, collectively called `oinit` [44, 45]. The architecture of the `oinit` solution is depicted in Fig. 6. It consists of three main components:

- **Certificate Authority (oinit-ca)**: a service that issues SSH certificates to users based on their federated identity, through integration with `motley-cue`. The CA can also be used to issue certificates for SSH servers, which mitigates the TOFU problem.

[4] https://ssh-oidc-web.vm.fedcloud.eu/.

- **Client-side tool (oinit)**: a client-side tool that automates the process of obtaining an SSH certificate from the CA, based on an AT.
- **SSH server-side tools (oinit-switch and oinit-shell)**: a tool that securely switches the user's identity from the service user to the one contained in the SSH certificate, and a custom, restricted shell for the service user.

Certificate Authority: oinit-ca

SSH certificates have been introduced in OpenSSH 5.3, and they are a more secure alternative to public key authentication. They are based on the same public key cryptography, but they are issued by a CA, which can ensure that they are bound to a user's identity, with an expiration date, and can be revoked by the CA. In addition, the CA can enforce the use of a passphrase, as well as other security policies.

SSH certificates contain a number of fields that provide additional information, which could be useful or relevant for our solution, such as:

- **Principals**: the user(s) or host(s) that the certificate is valid for.
- **Public key**: the public key that the certificate is signed for.
- **Signing CA**: the public key of the CA that signed the certificate.
- **Key ID**: an optional identifier for the certificate, which can be used for tracing, or in a key revocation list, optionally along with the **Serial** number.
- **Critical Options**: options that must be enforced by the server, such as the command that the user is allowed to run, or the accepted source addresses from which the certificate may be presented.
- **Extensions**: options that can be enforced by the server, such as X11 forwarding, agent forwarding, port forwarding, pty allocation, or user rc execution.
- **Validity**: the period of time during which the certificate is valid.

The crucial design decision for the oinit solution was concerned with the approach to be used to dynamically determine the Unix username on the SSH server (requirements R2 and R4–SSO and dynamic user provisioning). As explained with the PAM approach in Sect. "mccli", *sshd privilege separation* prevents the username from being changed during the authentication process. As such, the username must be known in advance. For this, the CA interfaces with `motley-cue` to: trigger user provisioning using token-based authentication, retrieve the username on the target system, and finally embed this information in the certificate. In the SSH certificate, the username will not only be set using the `Principals` field of the certificate, but also by making use of the `force-command` Critical-Option. The Principals field also contains a specific service user. This is to support the non-username operation of requirement R4. For this, the SSH certificates will use a specific service-user, which—after additional checks—executes the `force-command` to switch to the correct user. Since the correct username is also listed as a principal, both modes (ssh with and without username) will work. Figure 7 shows the contents for an example SSH certificate.

`oinit-ca` is implemented as an OIDC-protected REST API, which can be accessed by the client with an AT to obtain an SSH certificate. The service only supports HTTP and must be run behind an HTTPS reverse proxy to prevent snooping of access tokens, such as `nginx`. The CA relies on `motley-cue` for user authorisation and provisioning. The CA also provides an endpoint for requesting host certificates, which mitigates the TOFU problem since the SSH client can verify the host certificate with the CA.

SSH Server-Side Tools

The SSH server must be configured to support SSH certificates for authentication and trust the appropriate CAs. In addition, a service user (by default `oinit`) must be created, which will be used to log in to the server. The service user will then switch to the appropriate user as enforced by the certificate, using the *force-command* option.

```
$ ssh-keygen -L -f user-key-cert.pub
user-key-cert.pub:
    Type: ssh-ed25519-certv01@openssh.com user
          certificate
    Public key: ED25519-CERT SHA256:...
    Signing CA: ED25519 SHA256:...
    Key ID: "user@example.com"
    Serial: 0
    Valid: from 2024-11-20T00:00:00 to
        2024-11-21T00:00:00
    Principals:
        oinit
        diana
    Critical Options:
        force-command oinit-switch diana
    Extensions:
        permit-X11-forwarding
        permit-agent-forwarding
        permit-port-forwarding
        permit-pty
```

Fig. 7 Example of contents of an SSH certificate. The certificate is valid for the users `oinit` and `diana` until 2024-11-21, and it allows the user to only run the `oinit-switch` command, as well as to forward X11, agent, port, and pty

```
$ oinit add <ssh host address> [<CA URL>]
$ oinit add oinit-demo.vm.fedcloud.eu https
    ://oinit-demo.vm.fedcloud.eu
```

Fig. 8 Adding a new host to the list of `oinit`-enabled servers. The CA URL is optional and can be determined using DNS when missing

For user switching, we provide a tool called `oinit-switch`, which ensures secure switching of the user context. The tool relies on the Unix tool *su*, which was chosen over alternatives like *setuid* or *sudo* since it can also support, among others: other sources of user accounts (such as LDAP), aging account information (from `/etc/shadow` file), as well as user environment configurations (e.g. from `/etc/login.defs`). The tool also ensures that only the service user can switch to the given user. In addition, it provides support for executing a command as the user instead of starting an interactive shell.

An additional tool we provide for the server-side is `oinit-shell`, which is a custom, secure shell for the service user. The shell is used to restrict the allowed commands for the service user to `oinit-switch`, and to prevent interactive access for the service user. A detailed security analysis is given in [44].

Client-Side: oinit

Since our solution works without any change to existing SSH command lines, the user only needs a way to distinguish between `oinit`-enabled SSH servers and traditional ones, as well as a way to configure which CA to use for which SSH server. For this reason, we provide a client-side tool called `oinit`. It provides multiple subcommands that implement these configurations, as well as an automatic process of obtaining an SSH certificate from the CA.

In a preparation step for an SSH login, the user adds new hosts and their corresponding CAs to the list of servers that are `oinit`-enabled, as shown in Fig. 8. This configuration is stored in a user configuration file (`~/.ssh/oinit_hosts`), but system-wide configurations are also supported (`/etc/ssh/oinit_hosts`). The tool also provides other management subcommands, such as listing the configured hosts or removing a host.

The `add` command also adds additional required configurations for SSH commands to run seamlessly and fully transparent to the user, as shown in Fig. 9.

Once the host is added, the user can log in to the SSH server using the SSH command as usual. With the `match` subcommand, the `oinit` tool will automatically detect whether the host is `oinit`-enabled and perform the necessary steps to log in:

```
$ cat ~/.ssh/known_hosts
@cert-authority oinit-demo.vm.fedcloud.eu ssh
    -ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI...

$ cat ~/.ssh/config
# This 'Match' block was added by oinit.
#
# Please make sure it stays positioned on top
# of your ssh config file, assuring it will
# be applied before other 'Host' or 'Match'
# blocks that may interfere with oinit.
Match exec "oinit match %h %p"
    User oinit
... (remaining configuration)
```

Fig. 9 Example configurations added to the `~/.ssh/known_hosts` and `~/.ssh/config` files after adding a new host to the list of `oinit`-enabled servers

1. obtain an AT from `oidc-agent` for the selected OP.
2. generate a fresh SSH key pair to be used for the SSH certificate.
3. with the AT and the SSH key, obtain an SSH certificate from the CA.
4. store the private key and the certificate in the `ssh-agent`, for future uses until expiration.

The user will then be logged in to the server using the certificate. For the user, this process is seamless and does not require any additional steps (except selecting the OP on the first login, if it cannot be determined automatically), as shown in Fig. 10.

Usage Considerations

All the tools developed in this work are open-sourced under the MIT license. We provide packages for the most common Linux distributions in our repository,[5] as well as pre-release versions with the latest features.[6] We also provide client tools for Windows, via the Windows Subsystem for Linux (WSL), as well as a PuTTY plugin to enable SSH using the PAM-based approach [46]. The tools have been evaluated by multiple communities that use HPC or grid resources, such as the PUNCH4NFDI consortium [47] (providing infrastructure for particle, astroparticle, hadron, and nuclear physics research).

In this section, we address a number of considerations that should be taken into account when using the tools, such as limitations or use cases.

[5] https://repo.data.kit.edu.

[6] https://repo.data.kit.edu/prerel.

```
$ ssh oinit-demo.vm.fedcloud.eu
[1] https://aai-dev.egi.eu/auth/realms/egi
[2] https://aai.egi.eu/auth/realms/egi (Accounts: egi)
[3] https://accounts.google.com (Accounts: google)
[4] https://iam.deep-hybrid-datacloud.eu
[5] https://login-dev.helmholtz.de/oauth2 (Accounts: helmholtz-dev)
[6] https://login.helmholtz.de/oauth2 (Accounts: helmholtz, scope-max-aai)
[7] https://oidc.scc.kit.edu/auth/realms/kit (Accounts: kit)
[8] https://wlcg.cloud.cnaf.infn.it
? Please select a provider to use [1-8]: 2
✓ Received a certificate which is valid until 2024-11-22 14:41:13 +0100 CET
egi-eu_eosc-synergy-eu001@oinit-demo:~$
```

Fig. 10 Logging in to an `oinit`-enabled SSH server using the `ssh` command. When the server supports multiple OPs, the user is prompted to select the OP to use. Accounts configured in `oidc-agent` are shown in parentheses for each OP

Running `motley-cue` needs a port open to the outside world, since it has to be accessible not just by the PAM module, but also by the client-side tools (`mccli` and `webssh-oidc`). This can be a security risk, especially if the service is not properly secured. We recommend running `motley-cue` behind a reverse proxy, with SSL encryption, and with proper access control. The service should also be monitored for unusual activity, and logs should be kept for auditing purposes. The packages already provide helpful configurations for this purpose, e.g. for nginx as a reverse proxy, and a systemd service file.

`motley-cue` can also run on a host separate from the SSH server, depending on how the user database is accessed (for example, via LDAP). This enables a use-case where `motley-cue` is run on one dedicated host, and multiple SSH servers are run on separate hosts, which is typical in HPC environments. This relieves the SSH servers from the need to have a port open to the outside world, and allows the SSH host to be more secure, since it can be more easily monitored and maintained.

Alternatively, with the certificate-based solution, the CA can be the only component exposed to the outside world, while the `motley-cue` instance(s) can be run in a secure environment since it would only need to be accessible by the CA. This reduces the attack surface of the system, even in scenarios where there is a `motley-cue` instance per SSH server, e.g. when using local Unix accounts on each server.

The tools are designed to be secure and to follow best practices, but they are not immune to attacks. For example, the PAM module can be vulnerable to attacks that target the OIDC flow, such as token theft or token replay attacks. This can be mitigated by using short-lived tokens, and by using secure communication channels. The CA also provides a way to configure the validity of the certificates, as well as their revocation. For the certificate-based approach, we also provide mitigations through the server tools for several attacks, such as tty injection and interactive access for the service user [44].

Summary and Future Work

As a solution to the problem of federated identity-based SSH logins, we developed an ecosystem of tools that can be combined depending on the use-case and the requirements of the user or server administrator. The tools are designed to be flexible, secure, and user-friendly, and to provide a seamless experience for the user.

We provide both a PAM-based approach, which is suitable for most use-cases, and an SSH certificate-based approach, which is more in-line with standard SSH uses, but requires more setup. Both approaches rely on the same core component on the server-side, `motley-cue`, which provides authorisation, Unix account mapping and just-in-time provisioning, as well as a REST API for both server and client-side tools to interact with.

We also provide client-side tools that automate the process of obtaining the necessary tokens and certificates, and a web-based SSH client that allows users to log in without installing additional software.

Future work will include improvements on the client-side tools, such as: a non-interactive approach for the selection of the OP with the `oinit` tool, support for more advanced SSH command lines in the `mccli` tool, and support for `oinit` in the `webssh-oidc` tool. In addition, on the server-side we are planning to provide integrations with additional user management backends, such as the account linking service ALISE [48] or sssd, and to provide support for other notification methods in the approval mode of user provisioning, such as webhooks or messaging services. Furthermore, the packaging of the tools will be improved, so that the provided default configurations can ensure that the tools work together seamlessly with minimal configuration. Finally, we are planning to conduct a commercial security audit of all components of the ecosystem.

Author Contributions D.G. wrote the manuscript. All authors reviewed the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing Interests The authors declare no competing interests.

References

1. Ylonen T, Lonvick C (2006) The Secure Shell (SSH) Protocol Architecture. RFC Editor. https://doi.org/10.17487/RFC4251
2. OpenSSH: OpenSSH (2024) https://www.openssh.com Accessed 28 Nov 2024
3. Venkatachalam G (2007) The OpenSSH protocol under the hood. Linux J 156
4. Gasser O, Holz R, Carle G (2014) A deeper understanding of SSH: Results from Internet-wide scans. In: 2014 IEEE Network Operations and Management Symposium (NOMS), IEEE pp 1–9. https://doi.org/10.1109/NOMS.2014.6838249
5. Sakimura N, Bradley J, Jones M, Medeiros B, Mortimore C (2023) OpenID Connect Core 1.0 incorporating errata set 2. Technical report, OpenID Foundation. https://openid.net/specs/openid-connect-core-1_0.html
6. eduGAIN: eduGAIN. https://www.edugain.org Accessed 28 Nov 2024
7. AARC: The AARC policy development kit. https://aarc-community.org/policy-development-kit/ Accessed 28 Nov 2024
8. AARC Community members and AppInt members: AARC Blueprint Architecture 2019 (AARC-G045). 2019. https://doi.org/10.5281/zenodo.3672785
9. EGI Check-in. https://www.egi.eu/service/check-in-internal/ Accessed 27 Feb 2025
10. Helmholtz ID. https://hifis.net/aai Accessed 27 Feb 2025
11. Pashalidis A, Mitchell CJ (2003) A taxonomy of Single Sign-On systems. In: Information Security and Privacy: 8th Australasian Conference, ACISP 2003 Wollongong, Australia, July 9–11, 2003 Proceedings Springer, 8, pp 249–264. https://doi.org/10.1007/3-540-45067-X_22
12. Jones M, Bradley J, Sakimura N (2015) RFC7519: JSON Web Token (JWT). RFC Editor. http://www.rfc-editor.org/rfc/rfc7519.txt
13. Lonvick CM, Ylonen T (2006) The Secure Shell (SSH) Authentication Protocol. RFC Editor. https://doi.org/10.17487/RFC4252.
14. Cusack F, Forssen M (2006) RFC 4256: Generic message exchange authentication for the Secure Shell protocol (SSH). RFC Editor. https://www.rfc-editor.org/rfc/rfc4256.html
15. Hutzelman J, Salowey J, Galbraith J, Welch V (2006) RFC 4462: Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol. RFC Editor. https://doi.org/10.17487/RFC4462
16. Samar V. Unified login with Pluggable Authentication Modules (PAM). In: Proceedings of the 3rd ACM Conference on Computer and Communications Security, pp. 1–10 (1996)
17. Sermersheim J (2006) RFC 4511: Lightweight Directory Access Protocol (LDAP): The Protocol. RFC Editor. https://doi.org/10.17487/RFC4511 . https://www.rfc-editor.org/rfc/rfc4511.html
18. Moonshot: Moonshot Wiki 2024 https://moonshot-wiki.atlassian.net/ Accessed 28 Nov 2024
19. Rubens A, Rigney C, Willens S, Simpson WA (2000) Remote Authentication Dial In User Service (RADIUS). RFC Editor. https://doi.org/10.17487/RFC2865.
20. Hughes J, Maler E (2005) Security Assertion Markup Language (SAML) V2.0 Technical Overview. OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08 13, 12. Accessed 27 Feb 2025
21. Howlett J, Hartman S, Kourai K, Tsukada M (2016) RFC 7831: Application Bridging for Federated Access Beyond Web (ABFAB) Architecture. RFC Editor. https://doi.org/10.17487/RFC7831 . https://www.rfc-editor.org/rfc/rfc7831.html
22. National Center for Supercomputing Applications (NCSA): Grid Information Services SSH (GSI-SSH). http://grid.ncsa.illinois.edu/ssh/ Accessed 28 Nov 2024
23. Foster I, Kesselman C (1999) The globus toolkit. The grid: blueprint for a new computing infrastructure, 259–278
24. Alt J, Ananthakrishnan R, Chard K, Chard R, Foster I, Liming L, Tuecke S (2020) OAuth SSH with Globus Auth. PEARC '20, pp 34–40. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3311790.3396658
25. XSEDE: oauth-ssh. https://github.com/XSEDE/oauth-ssh Accessed 28 Nov 2024
26. Gao YA, Basney J, Withers A (2020) SciTokens SSH: Token-based authentication for remote login to scientific computing environments. In: Practice and Experience in Advanced Research Computing 2020: Catch the Wave. PEARC '20, . Association for Computing Machinery, New York, NY, USA, pp 465–468. https://doi.org/10.1145/3311790.3399613
27. Withers A, Bockelman B, Weitzel D, Duncan B, Gaynor J, Basney J, Tannenbaum T, Miller Z (2018) SciTokens: Capability-Based Secure Access to Remote Scientific Data. In: Proceedings of the Practice and Experience on Advanced Research Computing: Seamless Creativity. PEARC '18. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3219104.3219135
28. Basney J, Flanagan H, Fleury T, Gaynor J, Koranda S, Oshrin B (2019) CILogon: Enabling Federated Identity and Access Management for Scientific Collaborations. In: Proceedings of International Symposium on Grids & Clouds 2019 (ISGC2019), vol. 351, p. 031. https://doi.org/10.22323/1.351.0031
29. Surkont J, Prochazka M. PAM module for OAuth 2.0 Device Authorization Grant. https://github.com/ICS-MU/pam_oauth2_device Accessed 28 Nov 2024
30. Jenssen J, Dack T. PAM module for OAuth 2.0 Device flow. https://github.com/stfc/pam_oauth2_device Accessed 28 Nov 2024
31. IRIS: IRIS: Identity and Access Management for Research. https://iris.ac.uk Accessed 28 Nov 2024
32. Es M, Dijk N, Fokkinga F. Federated authentication for non-web-based research services. https://github.com/SURFscz/pam-weblogin Accessed 28 Nov 2024

33. Wolniwiecz P, Kaliszan D. PAM SSH for Pracelab. https://git.man.poznan.pl/stash/scm/pracelab/pam.git Accessed 28 Nov 2024
34. Petersen MF, Hald M, Coulouarn T. SSH Certificates in a Federated World. https://github.com/wayf-dk/ssh-certs-in-a-federated-world Accessed 28 Nov 2024
35. Gravitational Inc.: Teleport. https://goteleport.com/blog/how-to-configure-ssh-certificate-based-authentication/ Accessed 28 Nov 2024
36. Smallstep: Smallstep SSH. https://smallstep.com/sso-ssh/ Accessed 28 Nov 2024
37. Gudu, Diana: motley_cue. https://motley-cue.readthedocs.io/ Accessed 28 Nov 2024
38. Hardt, M.: FLAsk with Access Tokens (FLAAT). https://doi.org/10.5281/zenodo.7433610 .
39. FEUDAL Client Adapter. https://pypi.org/project/feudalAdapter/ Accessed 28 Nov 2024
40. Ramírez, Sebastián: FastAPI. https://fastapi.tiangolo.com/ Accessed 28 Nov 2024
41. Zachmann G. Oidc-agent. https://doi.org/10.5281/zenodo.4966816 .
42. pexpect. https://pexpect.readthedocs.io/ Accessed 28 Nov 2024
43. Gudu D mccli (motley_cue Command Line Interface). https://mccli.readthedocs.io
44. Brocke L (2023) Certificate-based OpenSSH for Federated Identities. Master's thesis, Karlsruher Institut für Technologie (KIT). https://doi.org/10.5445/IR/1000165236 . 46.21.02; LK 01
45. Brocke L oinit suite: Certificate-based OpenSSH for Federated Identities. https://codebase.helmholtz.cloud/m-team/oidc/ssh/oinit Accessed 28 Nov 2024
46. Keyboard-Interactive OIDC Plugin for PuTTY. https://codebase.helmholtz.cloud/m-team/oidc/ssh/ki-oidc-plugin Accessed 28 Nov 2024
47. Drabent A, Freyermuth O, Giffels M, Hoeft M, Künsemöller J, Roland B, Schwarz D, Wissing C (2024) Federated heterogeneous compute and storage infrastructure for the punch4nfdi consortium. EPJ Web of Conferences 295, 07020. https://doi.org/10.1051/epjconf/202429507020 . 51.11.01; LK 01
48. Hardt M. Account LInking SErvice (ALISE). https://doi.org/10.5281/zenodo.10262609