

OAuth for TeraGrid Gateways

Jim Basney <jbasney@ncsa.uiuc.edu>

Jeff Gaynor <jgaynor@ncsa.uiuc.edu>

Table of Contents

[Document Status](#)

[Summary](#)

[Background](#)

[TeraGrid Science Gateways](#)

[TGUP InCommon \(Shibboleth\) Support](#)

[Unvetted and Vetted TGUP Accounts](#)

[CILogon](#)

[OAuth](#)

[TGUP OAuth Protocol Overview](#)

[Design and Implementation](#)

[TGUP \(OAuth Server\)](#)

[OAuth Server Data](#)

[OAuth Client Table \(oauth.clients\)](#)

[OAuth Client Approval Table \(oauth.client_approvals\)](#)

[OAuth Transaction Table \(oauth.transactions\)](#)

[Science Gateways \(OAuth Client\)](#)

[MyProxy](#)

[Project Plan](#)

[Security Considerations](#)

[Operational Considerations](#)

[Issues/Questions](#)

[TGUP OAuth Protocol Specification](#)

[OAuth Endpoints](#)

[OAuth Signature Method](#)

[Message Specification](#)

[Temporary Credential Request: Request](#)

[Temporary Credential Request: Response](#)

[Resource Owner Authorization](#)

[Callback](#)

[Token Request](#)

[Token Request: Response](#)

[Get Certificate: Request](#)

[Get Certificate: Response](#)

Document Status

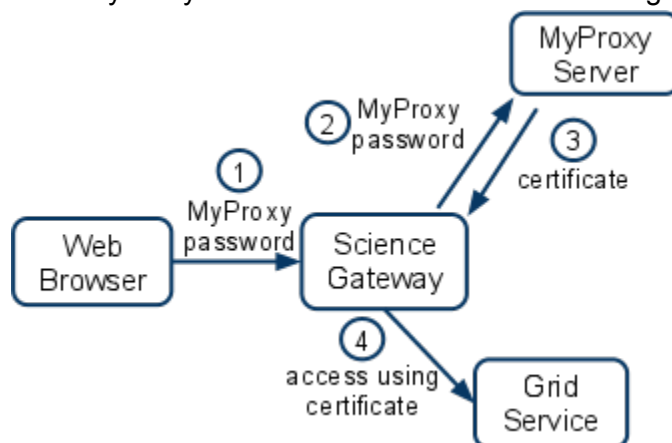
This document is a DRAFT. Please send comments to the authors.

Reviews and approvals are summarized in the following table:

Group	Received Comments?	Approved By	Approval Date
Gateways		Nancy Wilkins-Diehr	Pending
Globus Online	Mar 21 2011	Rachana Ananthakrishnan	Pending
Operations	Mar 25 2011	Jeff Koerner	Pending
Portals	Mar 25 2011	Maytal Dahan	Pending
RP Forum		John Towns	Pending
Security		Jim Marsteller	Pending
TGCDB		Leonard Carson	Pending
User Services		Sergiu Sanielevici	Pending

Summary

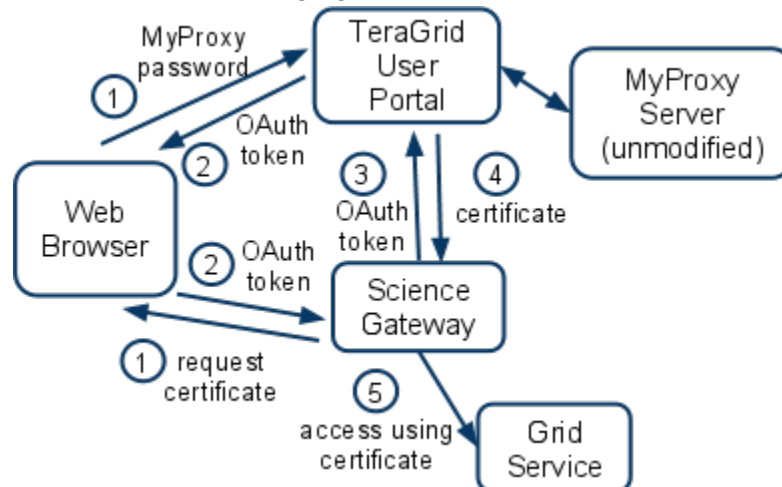
Currently, when a TeraGrid science gateway allows a researcher to use his or her own TeraGrid account, *rather than using a community account*, for access to TeraGrid resources, the science gateway uses the TeraGrid MyProxy server as illustrated in the following figure:



The researcher provides his or her TeraGrid MyProxy username and password to the science gateway, which the gateway uses to obtain a (short-lived) certificate for the researcher from the

TeraGrid MyProxy server, so the gateway can access TeraGrid resources on the researcher's behalf, by authenticating with the researcher's certificate. A significant drawback to this approach is that it discloses the researcher's (typically long-lived) TeraGrid MyProxy password to the science gateway.

To address this drawback, TeraGrid will provide an OAuth service, integrated with the [TeraGrid User Portal](#), as illustrated in the following figure:



When the science gateway requires a certificate to act on the researcher's behalf, it redirects the researcher's browser to the TeraGrid User Portal (TGUP), where the researcher authenticates (as usual) and approves (or denies) use of his or her TeraGrid credentials by the science gateway. If the researcher successfully authenticates and approves the use, the TGUP redirects the researcher's browser back to the science gateway with a one-time-use OAuth token that the gateway uses to obtain a certificate from the TGUP OAuth service, so the gateway can access TeraGrid resources on the researcher's behalf. The TGUP OAuth service will implement the OAuth 1.0a protocol, which is not shown in full detail above, but is specified in [RFC 5849](#). The TGUP OAuth service will require registration of science gateways and only allow authenticated requests from registered gateways.

The result is that TG researchers only enter their TGUP password on the TGUP, rather than also at science gateway web sites.

Note that this approach applies to web browser interactions and does not impact use of *myproxy-logon* on the command-line.

Note also that this approach is not a replacement for the widely adopted [community account](#) model for science gateways, which enables gateways to serve users who don't themselves have TeraGrid accounts. *TeraGrid science gateways using the community account model do not need to use this OAuth service.* The OAuth service is only for cases where gateways want to support authentication using individual TeraGrid accounts, as an alternative to the community account model.

Background

In this section we review background material that will inform the design and development of the TGUP OAuth service.

TeraGrid Science Gateways

The TGUP OAuth service supports TeraGrid science gateways. Visit the [TeraGrid Science Gateways](#) home page for background materials, and the [Science Gateways Use Cases](#) page for details about current gateway implementations.

TGUP InCommon (Shibboleth) Support

The effort to develop and deploy the TGUP OAuth service will be similar in some ways to our previous effort to develop and deploy the TGUP InCommon (Shibboleth) capability. Please refer to the [TGUP Shibboleth documentation](#) for design, implementation, and support materials related to that effort. See also our [IDtrust paper](#) (and [slides](#)) for a description of TeraGrid Single Sign-On (SSO) and how it is integrated with InCommon authentication.

Unvetted and Vetted TGUP Accounts

The [design](#) and [implementation](#) documents for the TGUP “Unvetted/Vetted” account management process also provide a reference for the TGUP OAuth work, both as an example of how TGUP changes are developed and deployed and as well as providing documentation for TGUP account management processes that the TGUP OAuth service must integrate with.

CILogon

The CILogon Service (<https://cilogon.org/>) provides an example of using OAuth for certificate issuance to portals. See the [CILogon Portal Delegation](#) page for more details, and visit <https://demo.cilogon.org> for a demonstration. TeraGrid will be modifying existing code from the CILogon project for this work.

OAuth

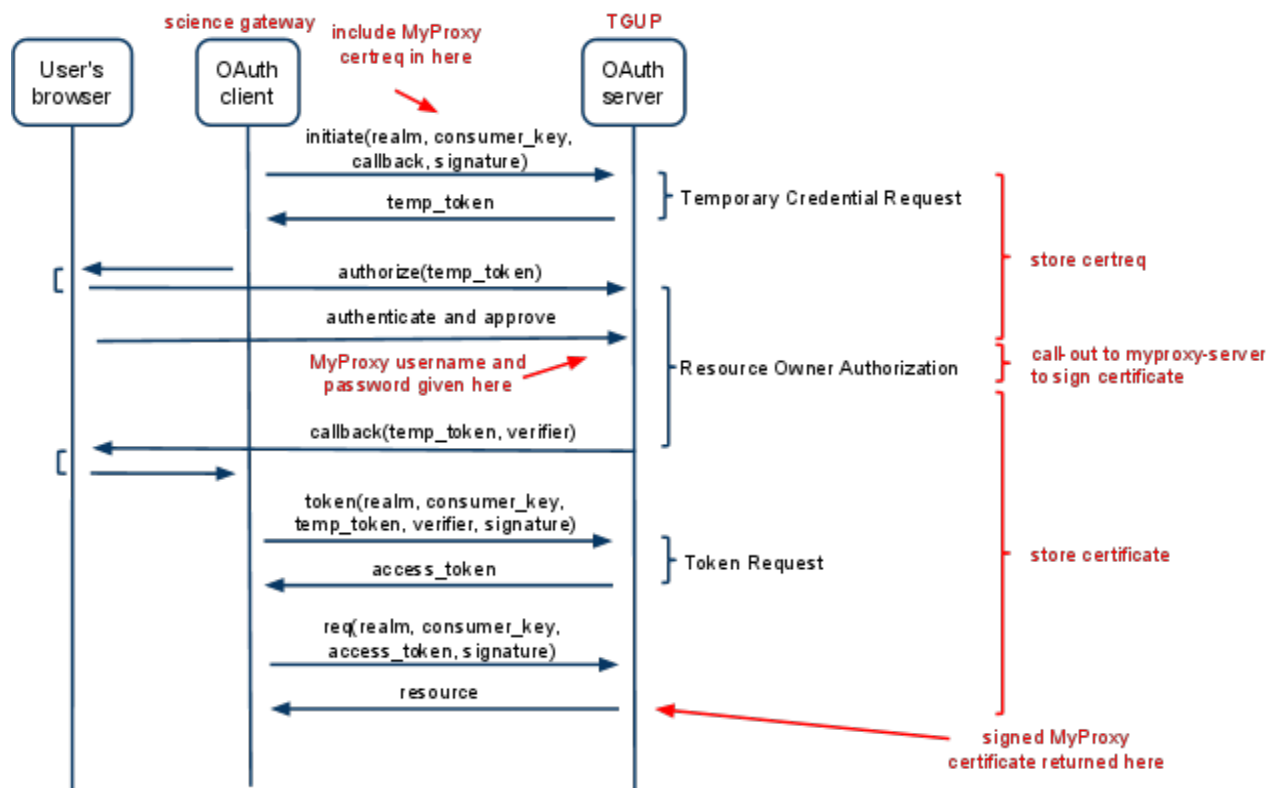
OAuth provides a relatively simple yet powerful mechanism for resource owners (that is, end users) to allow controlled, third-party access to their resources. This requirement is common across many applications, including the science gateways and web portals that are the focus of this work. In OAuth, the resource owner interacts with an OAuth server to approve access to the resource by the third party (the OAuth client) and issue tokens to the third party for secure access to the resource. The use of separate OAuth tokens enables the resource owner to allow

access without exposing his or her resource credentials (for example, passwords) to the third party. OAuth tokens can be for one-time use or renewable, can have limited lifetimes, and can limit the scope of access to the resource in various ways. The OAuth 1.0 protocol was initially developed in 2007 and was updated to version 1.0a in April 2009 to address a security flaw. OAuth 1.0a was published as RFC 5849 in April 2010. An update to the protocol, OAuth 2.0, is currently under development in the IETF OAuth working group (see [draft-ietf-oauth-v2](#)), and is expected to be finalized in 2011. Open Source OAuth 1.0 implementations are widely available, and OAuth 1.0 support is included in many Web frameworks.

OAuth is a very flexible security protocol, and its strength depends on how it is used. The management of OAuth keys and secrets and the methods employed for authentication and signing critically impact the overall security of the system. For example, the widely reported compromise of Twitter's use of OAuth in September 2010 was a result of weaknesses in how Twitter employed the OAuth protocol, rather than a vulnerability in OAuth itself (see [article](#)). As another example, the question of message signatures versus bearer tokens has been an active topic of discussion in the OAuth 2.0 working group, in part regarding trade-offs between implementation complexity and threat protection in different scenarios (see [article](#) and [draft-tschofenig-oauth-signature-thoughts](#)). The security profile of an OAuth use case depends on many factors including the type of resource being protected, contacting a well-known service endpoint versus doing dynamic service discovery, and whether the user agent is a web browser, native application, or autonomous service. Our use case here fits a widely adopted OAuth pattern: web browser access to a well-known service endpoint using message signatures.

TGUP OAuth Protocol Overview

The following figure presents the TGUP OAuth protocol flow:



As the figure illustrates, the TGUP OAuth server must support two interfaces: one for interaction with the user's browser, and the other for interaction with the science gateway. In OAuth 1.0a ([RFC 5849](#)) terminology, the TGUP acts as the *OAuth server*, the science gateway acts as the *OAuth client*, and the user (researcher) acts as the *OAuth resource owner*.

The science gateway (OAuth client) initiates the workflow by generating a private key for the user, submitting a certificate request to the OAuth server, and obtaining a "temporary token" from the OAuth server, then redirecting the user's browser (with the "temporary token") to the OAuth server. The user authenticates to the OAuth server (at <https://portal.teragrid.org/oauth>) using his or her TGUP (MyProxy) username and password and approves the gateway's request for credentials. The OAuth server then redirects the user's browser back to the science gateway (OAuth client) with a "verifier" that the gateway then uses in an OAuth token request to the OAuth server, followed by an OAuth resource request to obtain the certificate. All of these interactions are over HTTPS (HTTP Over TLS). Note that the user's private key is never sent over the network: it is generated locally at the science gateway, and the gateway sends only the public key in a certificate request to the OAuth server for signing by MyProxy.

The full protocol message specification is provided at the end of this document

Design and Implementation

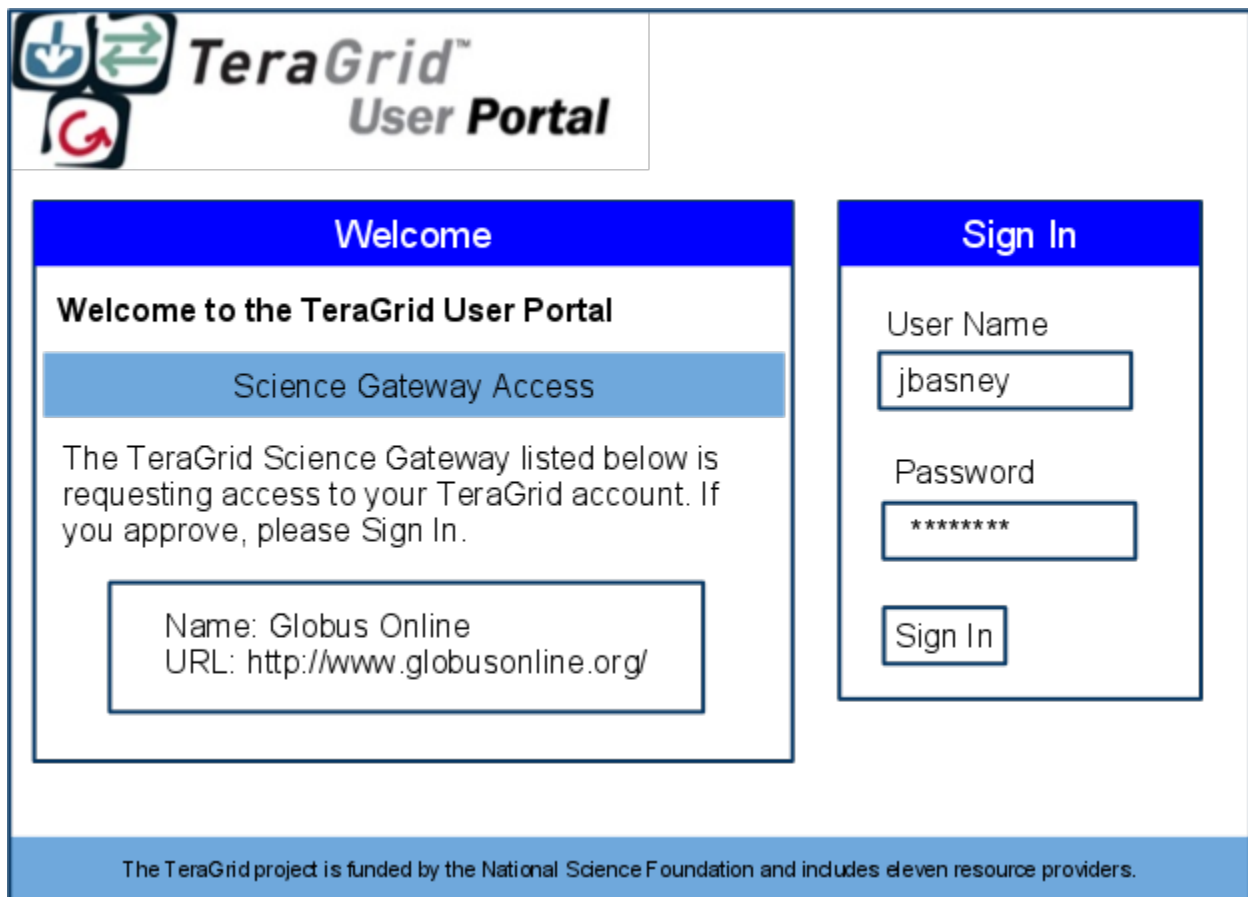
TGUP (OAuth Server)

For the TGUP component of the OAuth capability, we will develop an OAuth service (Tomcat Java servlet) that serves requests both from user browsers and from science gateways over HTTPS on port 443. The servlet's browser interface will support customization (CSS) to match the look-and-feel of existing TGUP components. It will run under <https://portal.teragrid.org/oauth> and will not require changes to the TGUP homepage at <https://portal.teragrid.org/>.

The OAuth service requires no direct integration with existing TGUP components and has no dependency on Liferay. Instead, it provides its own independent login page and OAuth session management (i.e., there will not be an "OAuth Sign In" option added to the TGUP homepage). However, the OAuth service is co-located with the TGUP to provide a consistent TeraGrid user experience (i.e., users log in at <https://portal.teragrid.org/>... URLs, whether they're using the TGUP or using OAuth with a science gateway). The servlet runs in the TGUP's Tomcat instance, behind the TGUP's Apache HTTPD in its own URL subdirectory (<https://portal.teragrid.org/oauth>). The HTTPD is responsible for SSL processing for the HTTPS connections (as with existing TGUP components).

The service has a "whitelist" of gateways (OAuth clients) that it serves, and it provides a web form for registering new OAuth clients. The service supports fail-over to multiple MyProxy servers (i.e., myproxy.teragrid.org and myproxy.psc.teragrid.org), similar to existing TGUP services.

A mock up of the TGUP OAuth Authorization page (<https://portal.teragrid.org/oauth/authorize>) follows:



The screenshot shows the TeraGrid User Portal interface. At the top left is the TeraGrid logo, which consists of three stylized icons (a blue arrow pointing down, a green arrow pointing right, and a red arrow pointing up) next to the text "TeraGrid™ User Portal". Below the logo, the page is divided into two main sections: "Welcome" and "Sign In".

The "Welcome" section has a blue header with the word "Welcome" in white. Below the header, it says "Welcome to the TeraGrid User Portal". Underneath, there is a light blue button labeled "Science Gateway Access". Below the button, a message states: "The TeraGrid Science Gateway listed below is requesting access to your TeraGrid account. If you approve, please Sign In." Below this message is a box containing the text: "Name: Globus Online" and "URL: <http://www.globusonline.org/>".

The "Sign In" section has a blue header with the words "Sign In" in white. Below the header, there are two input fields. The first is labeled "User Name" and contains the text "jbasney". The second is labeled "Password" and contains a series of asterisks "*****". Below the password field is a button labeled "Sign In".

At the bottom of the page, there is a blue footer bar with the text: "The TeraGrid project is funded by the National Science Foundation and includes eleven resource providers."

The authorization page serves two important functions: 1) explicitly obtain approval from the TeraGrid user for the gateway's use of the user's account, and 2) prompt the TeraGrid user for his or her TeraGrid username and password. The name and URL for the gateway making the request are read from the OAuth Client Table (see below), according to the information provided by the gateway at registration time. This page will only be shown for digitally signed OAuth requests by approved, registered gateways; in other cases, an error message will be displayed and the TeraGrid user will not be prompted for a password. Upon clicking the "Sign In" button, the TeraGrid user's browser is redirected back to the gateway, so in normal operation, this is the only <https://portal.teragrid.org> page the user will see during an OAuth transaction.

OAuth Server Data

The TGUP OAuth server is a stateful service. It must store registration records for OAuth clients (science gateways), which are updated infrequently, and must also store OAuth session data for each protocol transaction in progress. This requires three tables: the OAuth Client Table, the OAuth Client Approval Table, and the OAuth Transaction Table, all of which reside in the same schema, named `oauth`. We propose to store this schema and all these tables in the TeraGrid Central Database (TGADB), with corresponding users who are granted specific access and with a separate administration account for these tables. The data stored in the (fully qualified) tables:

OAuth Client Table (oauth.clients)

Column name	Type	Comments
oauth_consumer_key+	character varying	The identifier portion of the OAuth client credentials, i.e., a identifier for the gateway.
oauth_client_pubkey	character varying	The OAuth client (gateway) public key.
name	character varying	A name for the OAuth client (gateway) for display to the user.
home_url	character varying	The home page URL for the OAuth client (gateway) for display to the user.
creation_ts	timestamp	A timestamp entry for when the entry was created.
error_url	character varying	A URL for display to the user that the user can visit for assistance in case of errors.
email	character varying	OAuth client (gateway) email address. Used for notification when the client (gateway) is approved.

OAuth Client Approval Table (oauth.client_approvals)

Column name	Type	Comments
oauth_consumer_key+&	character varying	The identifier portion of the OAuth client credentials, i.e., a identifier for the gateway.
approved	boolean	A flag to indicate if this client (gateway) is approved. Set to false when the record is created. OAuth transactions are denied if it is false. Set to true after review by TeraGrid staff.
approval_ts	timestamp	A timestamp entry for when the client was approved.
approver	character varying	The username of the TG staff person who approved the client (gateway).

We store the approved flag in a separate table because it is controlled by a different webapp with different permissions.

OAuth Transaction Table (oauth.transactions)

Column name	Type	Comments
-------------	------	----------

temp_token+	character varying	The OAuth temporary credentials identifier, including a timestamp for enforcing a limited lifetime on transactions. This is the oauth_token returned in the temporary credential request.
temp_token_valid	boolean	A flag to indicate if the temp_token_secret is valid (i.e., has not yet been used). Once the temp_token_secret is used, it is marked invalid and can not be used again.
oauth_callback	character varying	An absolute URL back to which the server will redirect the resource owner when the resource owner authorization step is completed.
certreq	character varying	The certificate request submitted by the OAuth client (gateway) in the temporary credential request.
oauth_consumer_key&	character varying	The OAuth client identifier, i.e., the primary key for the associated OAuth Client Table record.
oauth_verifier	character varying	The OAuth verification code sent from the OAuth server to the OAuth client via the user agent during resource owner authorization.
access_token*	character varying	The OAuth access token identifier. This is the oauth_token returned in the OAuth token request.
access_token_valid	boolean	A flag to indicate if the access_token_secret is valid (i.e., has not yet been used). Once the access_token_secret is used, it is marked invalid and can not be used again.
certificate	character varying	The certificate issued by MyProxy to be returned to the OAuth client (gateway) by the OAuth server when a valid access token is presented.

+ = primary key

& = foreign key

* = indexed

Science Gateways (OAuth Client)

For the science gateway component of the OAuth capability, we will provide client libraries that science gateway developers can integrate with. Since we use the standard OAuth 1.0a protocol, we can use existing [OAuth client implementations](#). We will provide Java and Python client libraries.

As with the [CILogon OAuth Client API](#), the TeraGrid OAuth Client API will provide a very simple interface for science gateways. The gateway first calls a `requestCredential()` function, that initiates the transaction with the TGUP OAuth server and returns a URL for redirecting the user's browser to the TGUP OAuth service. The gateway then redirects the user's browser to the provided URL, where the user authenticates at the TGUP OAuth service, and then the TGUP OAuth service redirects the user back to the gateway's registered callback URL. When the user returns to the gateway, the gateway makes a second API call, to the `getCredential()` function, which completes the transaction with the TGUP OAuth server and returns the X.509 credential for the user. The API handles generation of private keys and certificate requests for the gateway as part of the protocol.

Each science gateway must complete the OAuth client registration process via a TGUP web form and be approved for TeraGrid OAuth access before using the TeraGrid OAuth service. The registration process is:

- A science gateway operator submits the OAuth registration form providing the following information:
 - **Science Gateway Name:** A friendly name for the science gateway for display to users on the TGUP OAuth Authorization page (see above).
 - **Science Gateway Home URL:** The home page URL for the science gateway for display to users on the TGUP OAuth Authorization page.
 - **Science Gateway URL:** A gateway-specific URL for display to the user in case of problems that tells the user how to get assistance from the gateway operators.
 - **Science Gateway Email Address:** An email address for notifying the science gateway operator when the gateway is approved for OAuth client access and when any other operational notices are needed.
 - **Science Gateway RSA Public Key:** A 2048 bit RSA public key corresponding to the RSA private key that the science gateway will use to sign OAuth messages.
- The form output includes the `oauth_consumer_key` (OAuth client identifier) generated by the OAuth server for this new OAuth client. The science gateway operator will need to configure the OAuth client software to use this `oauth_consumer_key` value when interacting with the TG OAuth server.
- TG staff (Science Gateways Area Director) are notified and review the submission.
- If the submission is approved, a TG staff person runs an application to update the "approved" flag in the OAuth Client Table (see below) and send a notification to the registered science gateway email address.
- The science gateway can now submit OAuth transactions.

Gateway operators can use the following commands to generate their RSA public/private keys for use with OAuth:

```
openssl genrsa -out oauth-privkey.pem 2048
chmod 0600 oauth-privkey.pem
openssl rsa -in oauth-privkey.pem -pubout -out oauth-pubkey.pem
```

They then configure the OAuth client software to use the `oauth-privkey.pem` file (and keep that file private/secure) and submit the contents of the `oauth-pubkey.pem` file in the OAuth registration form.

MyProxy

We do not expect to require any changes to the TeraGrid MyProxy servers (`myproxy.teragrid.org` and `myproxy.psc.teragrid.org`) to support this service. The MyProxy servers will continue to accept TeraGrid password authentication from the TGUP for issuance of certificates with lifetimes up to 11 days (264 hours).

Project Plan

1. [Target: March 2011] Design Complete. Sign-offs obtained on this document.
2. [Target: May 2011] Software Complete. OAuth client libraries and Java service implemented (by NCSA).
3. [Target: July 2011] TGUP Integration Complete.
 - a. Acceptance Test with Globus Online (using staging/test TGUP instance).
 - b. Consistent user interface achieved.
 - c. Gateway approval app deployed (developed by TGUP team).
 - d. Roll-out to production. Official announcement to gateways.

Security Considerations

1. All interactions with the OAuth service are over HTTPS (port 443), over an encrypted TLS channel with server authentication (using a `portal.teragrid.org` HTTPS certificate).
2. Science gateways generate private keys locally and never send them over the network. A science gateway sends a certificate request to the TGUP OAuth service to obtain a signed certificate. The TGUP never sees the user private keys.
3. TeraGrid MyProxy servers issue certificates valid for up to 11 days (264 hours), consistent with the [EUGridPMA Guidelines on Private Key Protection](#) and [NCSA CA Policies](#).
4. The gateway user enters his or her TGUP password via a web browser only on the TGUP login page (i.e., with a consistent TGUP look-and-feel and an `https://portal.teragrid.org` URL) over an encrypted TLS channel with server authentication. The TGUP sends the password for verification to the TeraGrid MyProxy server over an encrypted TLS channel with server authentication. Science gateways never see TGUP passwords.
5. The gateway user approves each certificate issuance by clicking a button on the TGUP.
6. OAuth clients (science gateways) must protect their credentials (i.e., private key associated with `oauth_client_pubkey`) (see [RFC 5849](#) Section 4.6). If compromise of the

OAuth client credentials is detected, access may be revoked by setting the approved flag to false in the OAuth Client Table.

7. The TGUP OAuth service uses the RSA-SHA1 signature method specified in [RFC 5849](#) Section 3.4. This method avoids the plaintext storage of credentials by the OAuth server (see [RFC 5849](#) Section 4.5), by using asymmetric cryptography (needing only a public key to be stored) instead of symmetric cryptography (shared secret).
8. See also the Security Considerations section of [RFC 5849](#) (OAuth 1.0a Protocol).
9. The OAuth service logs all successful and unsuccessful authentication attempts using log4j to the TGUP Tomcat logs. Log entries include: timestamp, browser IP address, TG username, OAuth client (gateway) identifier, and OAuth client (gateway) IP address.
10. In case of reported compromise of a TeraGrid password, current standard practice includes revocation of any currently valid certificates issued for the compromised TeraGrid account by TeraGrid MyProxy servers.

Operational Considerations

1. The TGUP OAuth service depends on the TeraGrid MyProxy service, which is replicated at NCSA (myproxy.teragrid.org) and PSC (myproxy.psc.teragrid.org). The TeraGrid MyProxy service depends on the TeraGrid Kerberos service (for verifying TeraGrid usernames and passwords), which is replicated at NCSA and PSC.
2. The TGUP OAuth service depends on the TGCDB.
3. The TGUP OAuth service is replicated according to the current TGUP replication method (one production instance, two warm backups, load balancer in front). It runs under the same Tomcat instance as Liferay.
4. A workflow is required for registration and approval of OAuth clients (gateways). Who should be notified of OAuth client registration requests (likely some combination of TG security, TGUP staff, and TG gateway coordinator)? Who should review and approve the requests? Who should have the ability to undo approvals (i.e., likely the security-wg will need this capability for incident response) to disable access by an OAuth client?
5. The TGUP OAuth service will be monitored by Nagios and Inca by doing periodic HTTPS GETs to <https://portal.teragrid.org/oauth/initiate> and <https://portal.teragrid.org/oauth/register>.

Issues/Questions

1. Does the TGUP OAuth service support InCommon authentication or only TGUP password authentication? Supporting only TGUP password authentication significantly simplifies the design. **Decision:** Support only TGUP password authentication for now, until we gain more experience with TGUP InCommon authentication.
2. Will the TGUP OAuth service run on the “secure server” or on the regular TGUP instance(s)? No motivation for running on the TGUP “secure server” has yet been identified. The TGUP OAuth service behaves very similarly to the Liferay instance(s), i.e.,

accepts TGUP passwords and obtains certificates from MyProxy. **Decision:** Run on the regular TGUP instances (primary and backups).

3. What data store will the TGUP OAuth service use? The TGCDB is a candidate. Should the same data store be used for long-term data (OAuth Client Table) as well as session data (OAuth Transaction Table)? Should session data be stored only in memory (i.e., will the OAuth service be replicated)? **Decision:** Store all data in the TGCDB to facilitate replication / fail-over.
4. Should sessions be shared between the OAuth service and Liferay? In other words, if a user logs in at the OAuth service, should that also log them in to Liferay (and vice versa)? This would be complicated to implement and is likely not worth the effort or complexity. **Decision:** Avoid all dependencies on Liferay, so do not share session information between OAuth and Liferay. The TGUP Sign In and TG OAuth Sign In pages will be clearly differentiated and will not reference each other or link to each other, so as to minimize user confusion.
5. Is the OAuth client whitelist required to restrict access to only approved TeraGrid Science Gateways? If yes, who will be responsible for approving additions to the whitelist? Preliminary decision: Require gateway registration and explicit approval by the Science Gateways Area Director (i.e., Nancy).
6. Is the OAuth service replicated? This has implications for data storage (Issue #3). **Decision:** Support replication consistent with the current TGUP deployment, using the TGCDB as shared storage for all OAuth state.
7. Should OAuth client (science gateway) IP addresses be registered and verified? [RFC 5849](#) Section 4.6 recommends registering OAuth client IP addresses and using IP addresses in addition to OAuth client credentials to verify client identities. This could add a significant management burden for the science gateways and TeraGrid staff to maintain, however. Preliminary decision: Address this issue by logging all access (with IP addresses in log messages) and monitoring access logs rather than requiring strict IP-based access control.
8. Which signature method should the OAuth service use? [RFC 5849](#) specifies three signature methods: HMAC-SHA1, RSA-SHA1, and PLAINTEXT. Both HMAC-SHA1 and PLAINTEXT are “shared secret” methods, meaning that they require a cleartext secret to be shared (and stored) by the OAuth client and server, bringing clear security risks (see [RFC 5849](#) Section 4.5). The RSA-SHA1 method uses asymmetric cryptography, so the OAuth server need only store the client’s public key to verify signatures, rather than generating and storing shared secrets. The likely drawbacks to the RSA-SHA1 method are: 1) asymmetric cryptography is slower and 2) generating and managing asymmetric keys can be more complex for clients. Preliminary decision: Use RSA-SHA1 to avoid the risks and complexity on the server-side of storing and generating shared secrets.
9. Should we have a webapp that lets users manage (view and revoke) their OAuth approvals? This is common for OAuth services that support long-lived approvals (e.g., connecting Facebook and Twitter accounts). In the case of the TGUP OAuth service, the user must explicitly sign in and approve each certificate issuance to a gateway (i.e., there is no “remember” checkbox). **Decision:** Because we require explicit user approval every time, there are no long-lived approvals to be viewed and revoked, so no webapp is

needed for this.

10. Should we support certificate renewal, i.e., allowing gateways to get another certificate for the user when the current user certificate nears expiration, without requiring user intervention? The TG OAuth service will issue certificates valid for up to 11 days (264 hours) to gateways. For long-running jobs, if some renewal facility is not provided, gateway users will need to sign in every 11 days so the gateway has a valid certificate for them. **Decision:** Certificate renewal adds significantly complexity, so we will not support it in the initial TG OAuth service.

TGUP OAuth Protocol Specification

Here we give the specification (with an example) for each protocol message.

OAuth Endpoints

Temporary Credential Request: <https://portal.teragrid.org/oauth/initiate>
Resource Owner Authorization: <https://portal.teragrid.org/oauth/authorize>
Token Request: <https://portal.teragrid.org/oauth/token>
Resource Request: <https://portal.teragrid.org/oauth/getcert>
Client Registration: <https://portal.teragrid.org/oauth/register>

OAuth Signature Method

The OAuth signature method is "RSA-SHA1" per [RFC 5849 Section 3.4.3](#).

Message Specification

All messages MUST be sent using HTTP Over TLS (HTTPS) per [RFC 2818](#).

All OAuth protocol parameters MUST be transmitted via HTTP request URI query according to [RFC 5849 Section 3.5.3](#) (i.e., URL encoded).

Temporary Credential Request: Request

Source: OAuth client

Target: OAuth server

Request: HTTPS GET to <https://portal.teragrid.org/oauth/initiate>

Note: Parameters may be in any order. Parameters not in the following table will be returned unaltered in the response. A request with duplicate parameters keys will be rejected. All parameters will be escaped as per the OAuth specification.

Parameter key	Parameter values	Comment
---------------	------------------	---------

oauth_signature_method	RSA-SHA1	Any other value is unsupported.
oauth_signature	computed signature	This is computed by the oauth library.
oauth_timestamp	current time in ms	This is computed by the oauth library.
oauth_nonce	integer value	This is computed by the oauth library.
oauth_version	1.0	This indicates OAuth 1.0 (RFC 5849).
oauth_consumer_key	string	Previously registered string that identifies the OAuth client.
oauth_callback	any valid HTTPS URL	The address to which the OAuth server will redirect the user's browser if authentication and authorization succeed. It MUST be HTTPS, not HTTP.
certreq	Base64 encoding of DER format PKCS#10 certificate request	This is the certificate request generated by the OAuth client for signing by the OAuth server. It MUST contain a 2048 bit RSA key. The signed certificate is returned in the final "getcert" call.

Example:

```
https://portal.teragrid.org/oauth/initiate?
oauth_consumer_key=dpf43f3p214k3l03&oauth_signature=74KNZJe
DHnMBp0EMJ9ZHt%2FXKycU%3D&oauth_signature_method=RSA-
SHA1&oauth_callback=https://portal.example.edu/oauth/
ready&oauth_timestamp=1273168086&oauth_nonce=8728518267508&oauth_versi
on=1.0&certreq=MIICVDCCATwCAQAwETEPMA0GA1UEAxMGaWdub3JlMIIBIjANBgkqhki
G9w0BAQEFAAOCAQ8AMIIB%0ACgKCAQEAAhUKrBs9%2B1GLUmfwjlunZVjud7%2Fnin0sdmO
YQHB2a2pBqdSQ3hG2wG0x9%2FNsst9AGiQ%2Fh%0AQ1LoR4uOARyBd8d5cJ7UOWN%2Btkt
kMovLgM4GcnCkVPsUEcA7ZrbbCLWzftgU25PuCdr1FF8xaahX%0AvR%2Bivcs%2FKKjAV4
UCTNI9ft%2Bwxg%2FE5JbQGpQZrIi8o%2B79MwzgdXvJfFVg0ZBDzNEB%2F7n10TYSW0Ez
%0AxmJABK6EouuaVmZCmLxoVfpwrn1%2BgfJPAPCWb27CXMCKt5zHmBKG7LbKD3hJhyZ25
MzZEu4R67eT%0AAEVuy5MaydhcW8rP%2FszGwZ5r%2B%2BtCFSCmNet9bwmaZkdlsQIDAQ
ABMA0GCSqGSIb3DQEBBQUAA4IB%0AAQBTu7r3D%2BUMHnb6JPYtdAdmep%2BxBFI21ws6Z
5rckvCAzAZSlcRxfGGZhrdgoOgfbE80FP1lhn1%2B%0Agvm13ku%2B4kCc1I8r7FwIOs7v
Cd2g%2B6gus%2BvgBM0hCxufuyNvRzbLtMbudj%2BPOReQMf0Y6%2Bng89DmW%0AXtm2J6Z
mpaQx3fNmJ8KFtuZdzdIjQhMg6772fKTDNOvThhtrXnch%2FWt%2BTg4jES0vWzFLL4OgF
bd1%0ADQX1HZXIoCjk%2BnVwTsPwm8E55p3qHKQ6fImN0%2BCPBjz%2F6PiqxpquR9kOkq
LNAXnbCob5XwMTM26P%0AC5WtKkEnFKAdCIDAS0Uv34fwZ%2BO7fHy2eestubpf
```


Temporary Credential Request: Response

Description: This is a standard HTTP response with code 200. Any other return code is to be treated as an error by the portal. The body of the response consists of a parameter list encoded as type application/x-www-form-urlencoded.

Source: OAuth server

Target: OAuth client

Body of response: oauth_token=value&oauth_callback_confirmed=true

Parameter key	Parameter value	Comment
oauth_token	string	The OAuth temporary credentials identifier.
oauth_callback_confirmed	true	Required for OAuth compliance.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded

oauth_token=hh5s93j4hdidpola&oauth_callback_confirmed=true
```

Note: The OAuth client must store the oauth_token for use in later messages. Because we are using the RSA-SHA1 method, no oauth_token_secret (shared secret) parameter is needed.

Resource Owner Authorization

Description: This standard HTTP response from the OAuth client to the user's browser redirects the user's browser to the OAuth server for authorization.

Source: OAuth client

Target: OAuth server

Access: HTTPS GET to <https://portal.teragrid.org/oauth/authorize>

Parameter key	Parameter value	Comment
oauth_token	string	The OAuth temporary credentials identifier.

Example: An example redirect URL is https://portal.teragrid.org/oauth/authorize?oauth_token=hh5s93j4hdidpola

Callback

Description: Redirect user's browser to the client's callback URL that was specified in the temporary credential request.

Source: OAuth server

Target: OAuth client

Request: HTTPS GET to specified callback URL

Parameter key	Parameter value	Comment
oauth_token	string	The OAuth temporary credentials identifier.
oauth_verifier	string	This new verification code is generated by the OAuth server.

Example:

`https://portal.example.edu/oauth/ready?`

`oauth_token=hdk48Djdsa&oauth_verifier=hfdp7dh39dks9884`

Token Request

Description: The OAuth client must exchange the now valid temporary credential for an access token.

Source: OAuth client

Target: OAuth server

Access: HTTPS GET to `https://portal.teragrid.org/oauth/token`

Parameter key	Parameter value	Comment
oauth_consumer_key	string	Previously registered string that identifies the OAuth client.
oauth_token	string	The OAuth temporary credentials identifier.
oauth_verifier	string	The verification code obtained from the callback.
oauth_signature_method	RSA-SHA1	Any other value is unsupported.
oauth_signature	computed signature	This is computed by the oauth library.
oauth_timestamp	current time in ms	This is computed by the oauth library.
oauth_nonce	integer value	This is computed by the oauth

		library.
oauth_version	string	This is currently equal to "1.0" to indicate OAuth 1.0.

Example:

```
https://portal.teragrid.org/oauth/token?
oauth_consumer_key=dpf43f3p214k3l03&oauth_token=hdk48Djdsa&
oauth_verifier=hfdp7dh39dks9884&oauth_signature=74KNZJeDHnM
Bp0EMJ9ZHt%2FXKYcU%3D&oauth_signature_method=RSA-
SHA1&oauth_timestamp=1273168086&oauth_nonce=8728518267508&oauth_versio
n=1.0
```

Token Request: Response

Description: This is a standard HTTP response with code 200. Any other return code is to be treated as an error by the portal. The body of the response consists of a parameter list encoded as type application/x-www-form-urlencoded.

Source: OAuth server

Target: OAuth client

Body of response: oauth_token=value

Parameter key	Parameter value	Comment
oauth_token	string	The OAuth access token identifier.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded

oauth_token=hh5s93j4hdidpola
```

Note: The OAuth client must store the oauth_token for use in later messages. Because we are using the RSA-SHA1 method, no oauth_token_secret (shared secret) parameter is needed.

Get Certificate: Request

Description: The OAuth client obtains the signed certificate.

Source: OAuth client

Target: OAuth server

Access: HTTPS GET to https://portal.teragrid.org/oauth/getcert

Parameter key	Parameter value	Comment
---------------	-----------------	---------

oauth_token	string	The OAuth access token identifier obtained in the Token Request.
oauth_consumer_key	string	Previously registered string that identifies the OAuth client.
oauth_signature_method	RSA-SHA1	Any other value is unsupported.
oauth_signature	computed signature	This is computed by the oauth library.
oauth_timestamp	current time in ms	This is computed by the oauth library.
oauth_nonce	integer value	This is computed by the oauth library.
oauth_version	string	This is currently equal to "1.0" to indicate OAuth 1.0.

Example:

```
https://portal.teragrid.org/oauth/getcert?
oauth_consumer_key=dpf43f3p2l4k3l03&oauth_token=hh5s93j4hdi
dpola&oauth_signature=74KNZJeDHnMBp0EMJ9ZHt%2FXKycU%3D&out
h_signature_method=RSA-
SHA1&oauth_timestamp=1273168086&oauth_nonce=8728518267508&oauth_versio
n=1.0
```

Get Certificate: Response

Description: A standard HTTP response. Any return code other than 200 indicates an error. The body of the response is the PEM encoded certificate with Content-Type: text/plain.

Source: OAuth server

Target: OAuth client

Example:

```
HTTP/1.1 200 OK
Content-Type: text/plain
```

```
-----BEGIN CERTIFICATE-----
```

```
MIIEIzCCAwugAwIBAgIDEbWMMMA0GCSqGSIb3DQEBBQUAMHsx CzAJBgNVBAYTA1VT
MTgwNgYDVQQKEy9OYXRpb25hbCBDZW50ZXI gZm9yIFNlcGVyY29tcHV0aW5nIEFw
cGxpY2F0aW9uczEgMB4GA1UEC xMXQ2VydGlmaWNhdGUgQXV0aG9yaXRpZXMxEDAO
BgNVBAMTB015UHJveHkwHhcNMTEwMzIxMTk0OTQ0WhcNMTEwMzIyMDc1NDQ0WjBc
MQswCQYDVQQGEwJVUzE4MDYGA1UEChMvTmF0aW9uYWwgQ2VudGVyIGZvc iBTdXB1
cmNvbXB1dGluZyBBcHBsaWNhdGlvbnMxEzARBgNVBAMTCkppbSBBCYXNuZXkwggEi
```

MA0GCSqGSib3DQEBAQUAA4IBDwAwggEKAoIBAQDMt7GjID/B68q5B7AUeJmnGWX9
0hE3yQu/OSKZpQopy39jjqoMZwBPTtt16eirdaHyYnvjj4ruvUcTd2y/TQrhdtfe
i1JvmyKliX7GiFP+m0q7ypDcBeetPbtneiNL3Hxjn4kZkX8bmILh5VKLdgtlvUhw
tGcVcAw6NccWYNwFIL3ge63804hMPOxggqVCskKbcOL4e5AZ4UPldCundGM3Z3yf
SktY/uDi2aIE/QWYb8AONvOVAott8wJLss4jXkJIxHSweNA6Q7AHeTY91p8V9t9b
cY7saCmWKqqKCP1XhsPhm5OgS2LwOhivuiDOZ+sikZqiBgnFmPV+lwJL5JdTAgMB
AAGjgc4wgcswDgYDVR0PAQH/BAQDAgSwMB0GA1UdDgQWBBSBONecg7MUYYDe5IA9
VE0NWctm3DAfBgNVHSMEGDAWgBTX/KUCdjr2E/oroeDmUDXHI8d7UTAMBgNVHRMB
Af8EAjAAMDQGA1UdIAQtMCswDAYKKwYBBAGkPmQCBTAMBgoqhkiG90wFAgIDMA0G
CyqGSib3TAUCAwIBMDUGA1UdHwQuMCwwKqAooCaGJGh0dHA6Ly9jYS5uY3NhLnVp
dWMuZWRR1L2YyZTg5ZmUzLmNybdANBgkqhkiG9w0BAQUFAAOCAQEAXwH09arINkBc
maUj2O5uEtFAkUt9XC5OmSZ4qoitgeeVbELFuZzAZn3NOKoNHVNV+Eg7FD8inIXN
AilKun8rofpORiEGr1RVgGx7fRnlvozSy4etQUkK/9O+U+whhXcD+oQGte5ryHJN
NwDst6viBM19Fn8cqvt3+b+Cv3+VoKdPdnG1SI1fZ3gIEcDkzqharvJ5jRoqClzY
f7fLhaIFSPnJy5cRX0Jff6xB/wfvbKTOGqDirRJYrNN4a/Ee+q0IFXz/L/fuZBjW
uWsRtpHV3CmxmTomORFxsQOj94HJDfTxbrEE/ol2irudZRLeY/tmwlcz6OV1Hkrc
O7PXvyR2Og==

-----END CERTIFICATE-----