G E N I

Global Environment for Network Innovations

Aggregate Manager API

Document ID: GENI-SE-CF-AMAPI-01.0

September 1, 2010

Prepared by:
The GENI Project Office
BBN Technologies
10 Moulton Street
Cambridge, MA 02138 USA

Issued under NSF Grant CNS-0741315

TABLE OF CONTENTS

1 Document Scope

This section describes this document's purpose, its context within the overall GENI document tree, the set of related documents, and this document's revision history.

1.1 Purpose of this Document

This document describes a programming interface, known as an API, for allowing GENI experimenters to contact aggregates of GENI resources to learn what resources are available and present requests for resource reservations. The GENI Aggregate Manager API is a common API for reserving disparate resources from multiple GENI aggregates. Prior to this API, each control framework specified a unique interface between aggregates and experimenters. The GENI Aggregate Manager API specifies a set of functions for reserving resources and describes a common format for certificates and credentials to enable compatibility across all aggregates in GENI.

1.2 Context for this Document

Figure 1-1 below shows the context for this document within GENI's overall document tree.

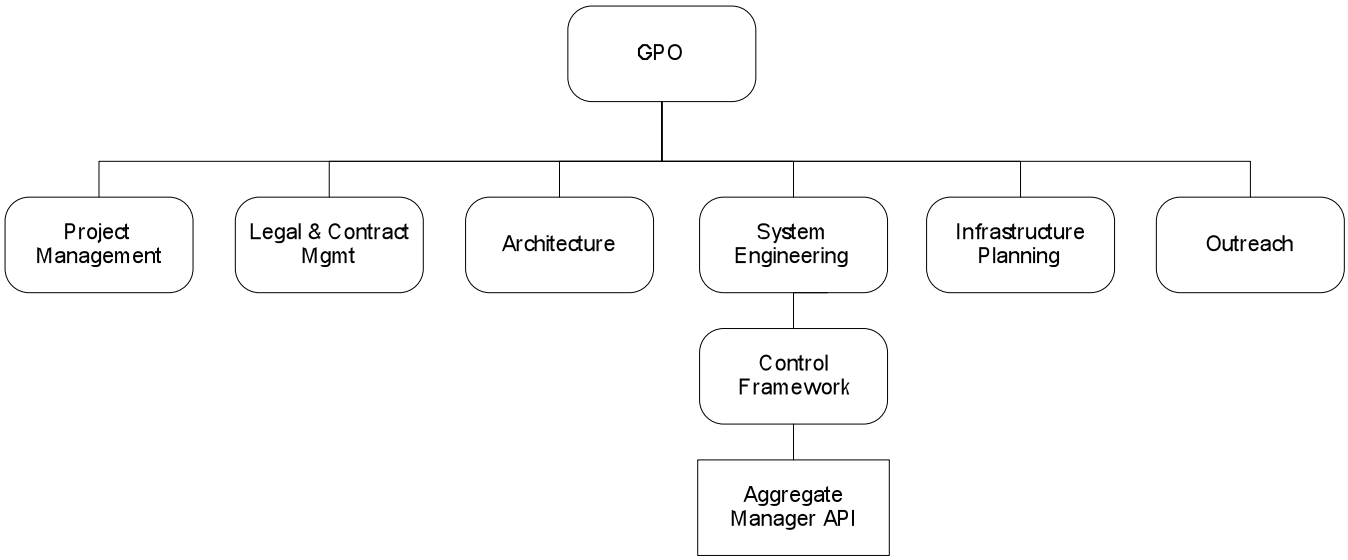

Figure 1-1. This Document within the GENI Document Tree.

1.3 Related Documents

The following documents of exact date listed are related to this document, and provide background information, requirements, etc., that are important for this document.

1.3.1 National Science Foundation (NSF) Documents

Document ID	Document Title and Issue Date
N / A	

1.3.2 GENI Documents

Document ID	Document Title and Issue Date
GENI-SE-SY-TS-UC-LC-01.0	Lifecycle of a GENI Experiment
GENI-SE-SY-RQ-01.9	GENI System Requirements (DRAFT)
GENI-SE-CF-RQ-01.3	GENI Control Framework Requirements (DRAFT)
GENI-SE-CF-PLGO-01.2	PlanetLab GENI Control Framework Overview (DRAFT)
GENI-SE-CF-PRGO-01.3	ProtoGENI Control Framework Overview (DRAFT)
GENI-SE-CF-ORGO-01.2	ORCA GENI Control Framework Overview (DRAFT)
GENI-SE-SY-SO-02.0	GENI System Overview
GENI-INF-PRO-S1-OV-1.12	GENI Spiral 1 Overview

1.3.3 Standards Documents

Document ID	Document Title and Issue Date
ISO 8601	Data elements and interchange formats – Information interchange – Representation of dates and times, December 2004
RFC 1421	Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures, February 1993
RFC 1950	ZLIB Compressed Data Format Specification version 3.3, May 1996
RFC 2818	HTTP Over TLS, May 2000
RFC 3151	A URN Namespace for Public Identifiers, August 2001
RFC 3280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, April 2002
RFC 3339	Date and Time on the Internet: Timestamps, July 2002
RFC 5246	The Transport Layer Security (TLS) Protocol Version 1.2, August 2008

1.3.4 Other Documents

Document ID	Document Title and Issue Date
SFA v1.0.1	Slice-Based Facility Architecture, August 8, 2008

1.4 Document Revision History

The following table provides the revision history for this document, summarizing the date at which it was revised, who revised it, and a brief summary of the changes. This list is maintained in chronological order so the earliest version comes first in the list.

Revision	Date	Revised By	Summary of Changes
01.0	9/1/10	T. Mitchell	Initial draft

2 GENI Overview

The Global Environment for Network Innovations (GENI) is a novel suite of infrastructure now being designed to support experimental research in network science and engineering.

This new research challenges us to understand networks broadly and at multiple layers of abstraction from the physical substrates through the architecture and protocols to networks of people, organizations, and societies. The intellectual space surrounding this challenge is highly interdisciplinary, ranging from new research in network and distributed system design to the theoretical underpinnings of network science, network policy and economics, societal values, and the dynamic interactions of the physical and social spheres with communications networks. Such research holds great promise for new knowledge about the structure, behavior, and dynamics of our most complex systems – networks of networks – with potentially huge social and economic impact.

As a concurrent activity, community planning for the suite of infrastructure that will support NetSE experiments has been underway since 2005. This suite is termed the Global Environment for Network Innovations (GENI). Although its specific requirements will evolve in response to the evolving NetSE research agenda, the infrastructure's conceptual design is now clear enough to support a first spiral of planning and prototyping. The core concepts for the suite of GENI infrastructure are as follows.

- **Programmability** – researchers may download software into GENI-compatible nodes to control how those nodes behave;
- **Virtualization and Other Forms of Resource Sharing** – whenever feasible, nodes implement virtual machines, which allow multiple researchers to simultaneously share the infrastructure; and each experiment runs within its own, isolated slice created end-to-end across the experiment's GENI resources;
- **Federation** – different parts of the GENI suite are owned and/or operated by different organizations, and the NSF portion of the GENI suite forms only a part of the overall 'ecosystem'; and
- **Slice-based Experimentation** – GENI experiments will be an interconnected set of reserved resources on platforms in diverse locations. Researchers will remotely discover, reserve, configure, program, debug, operate, manage, and teardown distributed systems established across parts of the GENI suite.

As envisioned in these community plans, the GENI suite will support a wide range of experimental protocols, and data dissemination techniques running over facilities such as fiber optics with next-generation optical switches, novel high-speed routers, city-wide experimental urban radio networks, high-end computational clusters, and sensor grids. The GENI suite is envisioned to be shared among a large number of individual, simultaneous experiments with extensive instrumentation that makes it easy to collect, analyze, and share real measurements.

3 Introduction

The GENI Aggregate Manager API is a common API for reserving disparate resources from multiple GENI aggregates. Prior to this API, each control framework specified a unique interface between aggregates and experimenters. The GENI Aggregate Manager API specifies a set of functions for reserving resources and describes a common format for certificates and credentials to enable compatibility across all aggregates in GENI.

This API has been implemented in multiple control frameworks, and will serve as the basis for ongoing integration among GENI control frameworks and tools. Using this document, new GENI-interoperable aggregate managers, tools, and clearinghouses may be built.

This API is based in large part on the Slice-Based Facility Architecture (SFA) version 1.0.1 [1]. Many of the concepts and terms used in this document are defined in the SFA document. Readers of this document are encouraged to also read that document, and its successors, for framing, background information and definitions.

This document describes the GENI Aggregate Manager API. Sections 4 and 5 provide background information and framing for the API. Section 6 describes the sample implementation of the API available for download and experimentation. Section 7 describes the API operations in detail and Section 8 describes common experimenter use cases. Section 9 provides an overview of how aggregates can join with other GENI services. Sections 10 - 13 describe the API building blocks in detail. These are the supporting elements that enable interoperability of aggregates, clients, and clearinghouses. The document closes with a brief discussion of possible future directions in Section 14.

4 Design

One goal of GENI is to facilitate national scale networking experiments that utilize resources across many administrative domains. As a first step, the GENI community has identified an initial API for aggregates that allows aggregates to federate with multiple clearinghouses and provides a common API between experimenters and aggregates.

The GENI Aggregate Manager API is a limited set of operations that were common between the PlanetLab [2] and ProtoGENI [3] projects. These particular frameworks were chosen because they were the most mature and they both exposed similar SFA derived interfaces. The API attempts to reconcile the different design decisions that PlanetLab and ProtoGENI made by specifying a minimal common set of operations that are sufficient to create meaningful experiments across aggregates. Further, the API specifies common certificate and credential formats to enable user authentication and privilege verification.

This document notes the key technical decisions made during the API's design. Our decisions were guided by the desire to follow common standards, allow for future growth, and to minimize the necessary changes to PlanetLab and ProtoGENI.

Figure 4-1 shows how we expect the API will be used in practice. GENI experimenters will interact with a software client. This client, in turn, communicates with a GENI clearinghouse to create a slice, receive a slice credential and learn what aggregates are available. The API between client and clearinghouse is not standardized as of this writing. A client must use the native protocol of the clearinghouse it is contacting. After creating a slice, the client software uses the GENI Aggregate Manager API to interact with aggregates on behalf of the experimenter. The experimenter, via the

client, can learn what resources are available, reserve a set of resources by creating a sliver, and return the resources to the available set by deleting the sliver when they are done.

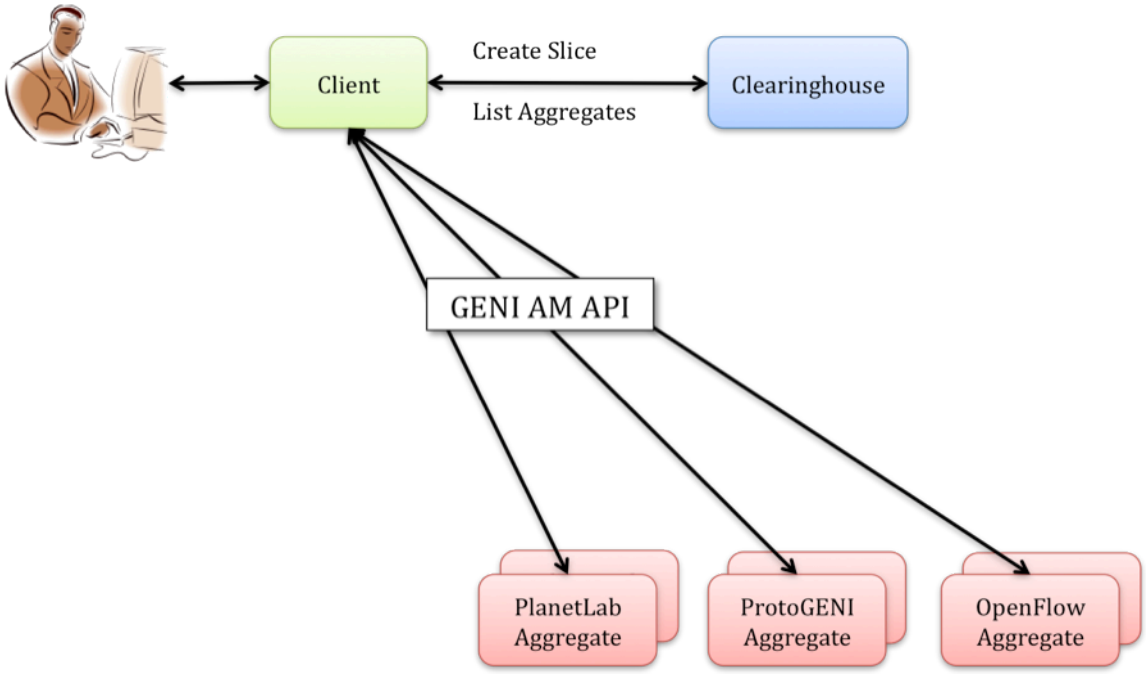

Figure 4-1. The GENI Aggregate Manager API in use.

4.1 SSL

GENI API implementations must use Secure Socket Layer (SSL) [RFC 5246] for authentication and encryption of all communications between clients and aggregates. Currently, only the client must be verified to the aggregate, and not vice-versa. SSL provides this identification in a broadly supported and readily implemented communications infrastructure.

It is important to resource providers that they be able to identify the experimenter who is responsible for the use of their resources. This identification forms the basis for tracing actions and auditing resource use. Because clients are acting on behalf of a human, they must pass a certificate that identifies that individual to the aggregate. It is preferred that all experimenters use certificates that are protected by a password to avoid inappropriate use of their GENI identity.

SSL was chosen as the authentication mechanism for GENI because of its broad availability, its relatively simple configuration, and its existing use in ProtoGENI and PlanetLab. Other authentications mechanisms, like Shibboleth [4], may be explored in future revisions.

4.2 XML-RPC

XML-RPC [5] over HTTPS [RFC 2818] is used as the invocation mechanism for GENI API operations. XML-RPC is a remote procedure call mechanism that uses HTTP for transport and XML as the encoding. XML-RPC is readily secured by SSL, can be used from a variety of programming languages and was already in use by both ProtoGENI and PlanetLab. These factors combined to make a compelling rationale for adopting XML-RPC over HTTPS as the invocation mechanism for the GENI API.

5 Relationship to SFA

The GENI Aggregate Manager API is intended to be compatible with the SFA. It also serves as a concrete implementation of a key interface within the SFA: the interface between experimenters and resources.

As the SFA continues to grow, we expect that this API will continue to track the SFA and grow with it. We also anticipate agreement upon additional GENI APIs based on other interfaces specified in the SFA. We believe that GENI will be one of many SFA-compatible network experimentation testbeds.

6 Sample API Software Implementation

The GENI Aggregate Manager API has been implemented at PlanetLab and ProtoGENI, and the source code for those implementations is available in those codebases. The GPO has also written a reference implementation of this API and an example client called Omni. This reference code can be used to further explain the semantics of the API, to test the interoperability of new tools and services, to provide examples of certificate and credential verification, and can be used as a framework for new development. This code is called the GENI Control Framework (gcf) and is available at http://www.geni.net (in the GENI API section of the GENI wiki).

This package, known as 'gcf', provides a simple Python [6] 2.6 implementation of the Aggregate Manager API. Developers can use gcf to understand in practice how to generate and validate URNs (Section 10), certificates (Section 11), and credentials (Section 12). See the included README.txt for dependencies and usage instructions.

The gcf codebase includes top-level documentation, optional Eclipse IDE project files, and a source tree. That source has 5 main scripts:

- *gen-certs.py* generates identity certificates
- *gam.py* runs the Reference Aggregate Manager
- *gch.py* runs the gcf test Clearinghouse
- *omni.py* is the Omni client
- *client.py* is a test script for testing API implementations

The library code is split into two python packages: ‘geni’ and 'sfa'. The SFA code is imported from the larger SFA library provided by PlanetLab. It includes code to create and verify certificates, credentials, and URNs. The geni package contains code written by the GPO which includes the servers, utility functions and the Omni client. Specific files that might be useful for reference include:

- *src/gen-certs.py* provides a single file that generates GENI compatible identity certificates for a clearinghouse, aggregate manager and user(s)
- *src/geni/am.py* implements the reference Aggregate Manager showing the semantics of the API operations, how credentials are verified and a trivial implementation of resource management via the API. The same file includes the AggregateManagerServer, which demonstrates how an XML-RPC server can be created in Python over SSL.
- *src/geni/ch.py* implements the reference Clearinghouse. It generates user and slice credentials for use with any GENI API compatible Aggregate Manager.

- *src/geni/util* is a Python package that contains helper functions to create and verify URNs, certificates, and credentials. It also includes an XML-RPC over SSL client.

The final component of gcf is the Omni client (*src/omni.py*). Omni demonstrates the use of the Aggregate Manager API from a client or experimenter perspective. It is a command-line tool that can obtain user and slice credentials from multiple frameworks (currently PlanetLab, ProtoGENI, and GCF) and subsequently interact with aggregates that support the GENI API. Omni communicates with clearinghouses via their native APIs because those protocols have not been standardized. A plug-in architecture allows future expansion to new clearinghouse APIs. Omni translates the RSpecs of each known aggregate type into a common language (omnispec) to simplify the display of resources to the user. Omni can be used by experimenters to reserve resources across multiple aggregates or by developers to test new implementations of this API. It can serve as an inspiration or as a starting point for more fully featured clients.

7 API Operations

7.1 Common elements

Many of the API operations have common parameters that are consistently used. For example, when operations target a slice, they accept a *slice_urn* argument. When operations require privileges they accept an array of credentials indicating the privileges that the experimenter has on the slice.

7.1.1 Slice URN

When API functions operate on a slice, a URN (a GENI identifier) is used to specify the slice. The format of URNs is detailed in Section 9. The slice URN is needed so that the aggregate can match the slice name against the provided slice credentials.

The use of the slice URN as an argument to API operations implies that all of the resources reserved within an aggregate are part of one sliver, and those resources are manipulated as a group, not individually. They can be renewed together and given up (deleted) together, but not separately. The Aggregate Manager API could be enhanced in the future to allow multiple slivers within an aggregate so that individual resources or distinct sets of resources could be acted upon independently.

7.1.2 Array of Credentials and semantics

Credentials denote a set of privileges that an actor has on a target object. For instance, a credential might grant Alice the control privilege on Slice Alpha. In this case, Alice is the actor, control is the privilege, and Slice Alpha is the target. Credentials are cryptographically signed so that they cannot be forged. See Section 12 for more information about credentials. In the API, credentials are formatted as strings and are passed in an array.

The API functions require that at least one credential in the array:

- Specifies the correct target (usually a slice)
- AND specifies the correct subject (usually the experimenter)
- AND contains appropriate privileges for the operation
- AND is signed by a valid authority for the target
- AND has not expired.

7.1.3 Return values

When API functions complete successfully they return values as documented below. All of the return types are standard XML-RPC types. When API functions encounter errors they return XML-RPC faults. These faults are generally handled through an exception handling mechanism on the client side. Clients of the Aggregate Manager API should handle these exceptions when invoking API operations.

7.2 GetVersion

Return the version of the GENI Aggregate Manager API supported by this aggregate.

```
struct GetVersion()
```

This operation is similar to ProtoGENI's GetVersion operation. The SFA specification does not include this operation.

7.2.1 Arguments

None.

7.2.2 Return value

The result is an XML-RPC struct with at least the following members:

```
{
  int geni_api;
}
```

`geni_api`

An integer indicating the revision of the Aggregate Manager API that an aggregate supports. The current version of the API is 1 (one).

Implementations can add additional members to the struct as desired. The prefix '*geni_*' is reserved for members that are part of this API specification. Implementations should choose an appropriate prefix to avoid conflicts. Candidates for addition to this struct include an RSpec version number for example.

7.3 ListResources

Return information about available resources, or return resources allocated to a slice.

```
string ListResources(string credentials[],
                     struct options)
```

This operation is similar to ProtoGENI's DiscoverResources operation and to the SFA's GetResources operation (sec. 6.2.4).

7.3.1 Arguments

`credentials[]`

An array of credentials. At least one credential must be valid for this operation (signed by a valid GENI certificate authority either directly or by chain, and not expired).

`options`

An XML-RPC struct containing members indicating the set of resources the caller is interested in or the format of the result. In addition to the members specified below, callers can pass additional members that specific aggregate manager implementations might honor. The prefix

'geni_' is reserved for members that are part of this API specification. Implementations should choose an appropriate prefix to avoid conflicts.

The following members are available for use in the options parameter. All aggregate managers are required to implement these options.

```
{
  boolean geni_available;
  boolean geni_compressed;
  string geni_slice_urn;
}
```

`geni_available`

An XML-RPC boolean value indicating whether the caller is interested in all resources or available resources. If this value is true, the result should contain only available resources. If this value is false both available and allocated resources should be returned. The Aggregate Manager is free to limit visibility of certain resources based on the credentials parameter.

`geni_compressed`

An XML-RPC boolean value indicating whether the caller would like the result to be compressed. If the value is true, the returned resource list will be compressed according to RFC 1950.

`geni_slice_urn`

An XML-RPC string indicating that the caller is interested in the set of resources allocated to the slice named by this URN. If no resources are allocated to the indicated slice by this aggregate, an empty RSPEC should be returned.

7.3.2 Return value

The return value is always a string. If *geni_compressed* is unspecified or set to false the return value will be an advertisement RSPEC in text format. If *geni_compressed* is specified and set to true the return value will be a string whose contents are base 64 encoded [RFC 1421] compressed advertisement RSPEC. Clients must first base 64 decode the string, then uncompress the decoded result.

7.4 CreateSliver

Allocate resources to a slice. This operation is expected to start the allocated resources asynchronously after the operation has successfully completed. Callers can check on the status of the resources using SliverStatus.

```
string CreateSliver(string slice_urn,
                    string credentials[],
                    string rspec,
                    struct users[])
```

This operation is similar to ProtoGENI's CreateSliver operation and to the SFA's CreateSlice operation (sec. 6.2.1).

7.4.1 Arguments

`slice_urn`

The URN of the slice to which the resources specified in the *rspec* argument will be allocated.

`credentials`

An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn*. Aggregates should ensure that the expiration time of the slice does not exceed the expiration time of the slice credential used to perform this operation.

`rspec`

A request RSPEC (plain text, uncompressed) containing the resources that the caller is requesting for allocation to the slice specified in *slice_urn*. These are expected to be based on resources returned by a previous invocation of ListResources.

`users`

An array of user structs, which contain information about the users that might login to the sliver that the AM needs to know about. Each struct must include the key *'keys'*, which is an array of strings and can be empty. The struct must also include the key *'urn'*, which is the user's URN string. The users array can be empty. For example:

```
[
  {
    urn: urn:publicid:IDN+geni.net:gcf+user+alice
   keys: [<ssh key>, <ssh key>]
  },
  {
    urn: urn:publicid:IDN+geni.net:gcf+user+bob
   keys: [<ssh key>]
  }
]
```

7.4.2 Return value

The return value is a manifest RSPEC indicating the resources that were allocated to the slice. The result RSPEC may contain additional information about the allocated resources.

7.5 DeleteSliver

Delete a sliver by stopping it if it is still running, and then deallocating the resources associated with it. This call will stop and deallocate all resources associated with the given slice URN.

```
boolean DeleteSliver(string slice_urn,
                     string credentials[])
```

This operation is similar to ProtoGENI's DeleteSliver operation and to the SFA's DeleteSlice operation (sec. 6.2.3).

7.5.1 Arguments

`slice_urn`

The URN of the slice whose sliver should be deleted.

`credentials`

An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn*.

7.5.2 Return value

Returns true on success and false on failure. If the result is failure the aggregate administrator should be notified so that the sliver's resources can be reclaimed administratively.

7.6 SliverStatus

Get the status of a sliver - from *configuring* to *ready* (for use) or (allocation) *failed*.

```
struct SliverStatus(string slice_urn,
                    string credentials[])
```

This operation is similar to ProtoGENI's SliverStatus operation. The SFA specification does not include this operation.

7.6.1 Arguments

`slice_urn`

The URN of the slice for which the sliver status is requested.

`credentials`

An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn*.

7.6.2 Return value

Returns an XML-RPC struct upon successful completion. The struct is of the following form:

```
{
  geni_urn: <sliver URN>
  geni_status: ready
  geni_resources: [ { geni_urn: <resource URN>
                      geni_status: ready
                      geni_error: ''},
                    { geni_urn: <resource URN>
                      geni_status: ready
                      geni_error: ''}
                  ]
}
```

The top level members of the returned struct pertain to the sliver as a whole. These members are:

`geni_urn`

The URN of the sliver as a string.

`geni_status`

A string indicating the status of the sliver. Possible values are shown in Table 7-1. More detailed information can be found in the value of the *geni_resources* member described below.

Value	Definition
configuring	At least one resource is being configured and none have failed.
ready	All resources in the sliver are ready.
failed	At least one resource in the sliver has failed.

unknown	The state of the sliver is not one of the known states.

Table 7-1: Sliver status values.

`geni_resources`

An array of structs. Each struct in the array gives the status of a resource in the sliver.

The members of the geni_resources struct(s) are as follows:

`geni_urn`

The URN of the resource as a string.

`geni_status`

A string indicating the status of the resource. Possible values are:

Value	Definition
configuring	The resource is being configured and is not yet ready for use.
ready	The resource is ready.
failed	The resource has failed.
unknown	The state of the resource is not one of the known states.

Table 7-2: Resource status values.

`geni_error`

A free form string. The aggregate manager should set this to a string that could be presented to a researcher to give more detailed information about the state of the resource if its status is failed.

7.7 RenewSliver

Renews the resources in a sliver, extending the lifetime of the slice.

```
boolean RenewSliver(string slice_urn,
                    string credentials[],
                    string expiration_time)
```

This operation is similar to ProtoGENI's RenewSlice operation. The SFA specification does not include this operation.

7.7.1 Arguments

`slice_urn`

The URN of the slice that is to have its sliver renewed.

`credentials`

An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn*.

`expiration_time`

A string in RFC 3339 format indicating the expiration_time desired by the caller. Note these times, per the RFC, must be in or relative to UTC. This time must be less than or equal to the slice duration in the slice credential. In other words, at least one supplied (slice) credential must still be valid at the desired new expiration time for this call to succeed.

7.7.2 Return value

Returns true on successful completion, false otherwise. It is assumed that the caller will have already extended the lifetime of the slice credential with the appropriate slice authority prior to calling RenewSliver.

7.8 Shutdown

Perform an emergency shut down of a sliver. This operation is intended for administrative use. The sliver is shut down but remains available for further forensics.

```
boolean Shutdown(string slice_urn,
                 string credentials[])
```

This operation is similar to ProtoGENI's Shutdown operation. The SFA specification does not include this operation.

7.8.1 Arguments

`slice_urn`

The URN of the slice is to have its sliver shut down.

`credentials`

An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn* or a valid administrative credential with sufficient privileges.

7.8.2 Return value

Returns true on success, false otherwise.

8 Use Cases

This section provides examples of how an experimenter might interact with the GENI API to perform typical tasks. The examples (provided below) assume that the experimenter has first contacted a GENI Clearinghouse and created a slice. A slice, in this context, is an empty container for slivers. No resources are requested or allocated when a slice is created. When an experimenter creates a slice she receives a slice credential. The slice credential indicates that the experimenter has been granted a set of privileges on the newly created slice. The slice also has an expiration date, encoded in the slice credential, indicating how long the slice is valid. See Section 11 for details on credentials.

8.1 Example 1: Sliver Creation

When an experimenter wants to allocate resources to her slice, she creates a sliver on a GENI aggregate. This task involves several GENI API operations:

1. The experimenter, via her client, invokes the **GetVersion** operation on the aggregate to determine what version of the API the aggregate implements. In the future, the result might dictate what other calls are possible on the aggregate.

2. The experimenter invokes **ListResources** to ask the aggregate what resources it has and which are available for use. The returned list is called an advertisement RSPEC and is in whatever format the aggregate chooses, but typically XML [7] or RDF [8]. Note that the result may be compressed. The experimenter will use this result to decide which available resources she wants to request.
3. The experimenter invokes **CreateSliver** to request a set of resources. The resource list passed with the invocation is called a request RSPEC, and is similarly in XML or RDF format, depending on the Aggregate. This operation returns a list of resources that were allocated to the sliver. The returned list is called a manifest RSPEC. This RSPEC tells the experimenter how her request was (or was not) fulfilled. The Aggregate may not have had as many nodes available as were requested, or may be able to specify particular machine names for the allocated nodes.
4. The experimenter invokes **SliverStatus** one or more times to determine the status of the resources that are part of the sliver. As the aggregate allocates and configures the resources, their status changes to Ready, indicating that the experimenter is free to begin using them, in whatever way the aggregate makes available. The API does not specify how the experimenter accesses or further configures the resources once they are allocated.

8.2 Example 2: Sliver Renewal

If the experimenter is not done with her experiment when the expiration date is approaching, she may want to renew her sliver. When an experimenter wants to extend the lifetime of a sliver associated with a slice she must first renew the slice with the clearinghouse. This operation should result in the experimenter receiving a slice credential with an updated and extended expiration date. It is then necessary to renew each sliver associated with the slice at each aggregate that contributes resources to the slice.

1. The experimenter invokes the **GetVersion** operation on the aggregate to determine what version of the API the aggregate implements (allowing her client to tailor future calls as needed).
2. The experimenter invokes the **RenewSliver** call with the slice_urn and new credentials to extend the lease on resources allocated to the sliver. The return value is true if the resources were successfully renewed. Then she can continue her experiment.

8.3 Example 3: Sliver Deletion

When an experimenter is finished with resources at an aggregate she can delete them from the slice. This makes those resources available to other experimenters.

1. The experimenter invokes the **GetVersion** operation on the aggregate to determine what version of the API the aggregate implements.
2. The experimenter invokes the **DeleteSliver** to relinquish the resources associated with a slice.

9 Federation

The GENI API facilitates the federation of GENI clearinghouses. Federated clearinghouses can share resources between their respective users. Federation is possible because of the standardized certificate and credential formats specified by this API, the use of the common API functions by experimenters to reserve resources from aggregates, and the explicit acceptance of federated root certificates by Aggregate Managers. Negotiating access policies remains a challenge that is beyond the scope of the API.

9.1 Mechanics

In order to federate with a clearinghouse an aggregate must share its XML-RPC endpoint (host name/IP address and TCP port) with the clearinghouse and must install the clearinghouse's root certificate to enable communication with its members. The XML-RPC endpoint is the hostname and port on which the XML-RPC server listens for connection requests. The clearinghouse is expected to publish a directory or listing of its federated aggregates.

Each clearinghouse is also expected to publish its root certificate. Aggregates that want to federate with a given clearinghouse must download and install the root certificate in order to accept and verify users and slice credentials issued by that clearinghouse. An aggregate is free to federate with any number of clearinghouses.

By installing a clearinghouse's root certificate an aggregate can communicate with all experimenters who possess a valid certificate issued by that clearinghouse. Aggregates implicitly trust whatever vetting process the clearinghouse performs on its members. Aggregates may choose to enforce additional local policy before fulfilling requests.

There is currently no established mechanism in the GENI API for distributing root certificates or for distributing certificate revocation lists.

10 Identifiers (URNs)

GENI identifies objects (be it a researcher, resource, clearinghouse, sliver, slice, or aggregate manager) are identified by a Uniform Resource Name (URN) [RFC 2141]. GENI URNs are encoded Public Identifiers [9], and thus follow RFC 3151, "A URN Namespace for Public Identifiers". The general format of a GENI URN is: `urn:publicid:IDN+<authority string>+<type>+<name>`. This format is adapted from the ProtoGENI URN format [10], which in turn is adapted from the GMOC GENI URN proposal [11]. All sections of the URN are mandatory. Note that additional '+' characters are allowed in the <name> section. URNs should be unique.

10.1.1 Public Identifiers

GENI URNs are in the URN namespace for Public Identifiers. As such, each GENI URN is of the form: urn:publicid:{transcribed-public-identifier}. RFC 3151 describes how public identifiers are transcribed to URNs (which involves collapsing whitespace and replacing certain characters with % encoded values).

The public identifier section of a GENI URN must begin with "IDN" to indicate that it uses an Internationalized Domain Name (see RFC 3151 for details). When transcribed, this means that all GENI URNs begin with "urn:publicid:IDN+".

To transcribe a Public Identifier to a URN use the following rules (as specified by RFC 3151):

From	Transcribe to
Leading and trailing whitespace	trim
whitespace	collapse to a single '+'
'//'	':'
'::'	';'
'+'	'%2B'
':'	'%3A'

'/'	'%2F'
';'	'%3B'
'"'	'%27'
'?'	'%3F'
'#'	'%23'
'%'	'%25

Table 10-1: Public identifier to URN translation.

10.1.2 Example translations:

Public ID	GENI URN
IDN plc//princeton authority sa	urn:publicid:IDN+plc:princeton+authority+sa
IDN gcf//gpo//gpolab user joe	urn:publicid:IDN+gcf:gpo:gpolab+user+joe
IDN gcf//gpo//gpolab node switch 1 port 2	urn:publicid:IDN+gcf:gpo:gpolab+node+switch+1+port+2

Table 10-2: Example URN Translations.

10.2 Usage

In the GENI API, URNs are used to name slices (as seen in arguments to GENI API calls), to identify users, and to label resources. URNs are also used in GENI certificates (to bind public keys to identifiers) and in Credentials (to grant permissions to source identifiers on target identifiers).

10.3 Authority String

Authority strings represent resource namespaces. For instance, 'plc' is the overall PlanetLab namespace and 'plc.princeton' and 'plc.bbn' are specific namespaces for the Princeton and BBN PlanetLab sites. The authority string format is 'toplevelauthority:sub_authority1:...:sub_authority_n'.

Only entities with URNs of type 'authority' are allowed to sign credentials for a namespace (except in the case of delegation).

For example, a ProtoGENI Clearinghouse with URN like 'protogeni:utah....' cannot issue a slice credential giving a user privileges on a PlanetLab slice (with urn 'plc:princeton....'). Only PlanetLab can grant rights over PlanetLab slices.

This means that all users and slices whose certificates are issued by a given Clearinghouse, will have a common format, eg:

`Urn:publicid:IDN+<Clearinghouse name>+<user or slice>+<username or slicename>`

Aggregate Managers with certificates issued by the same clearinghouse should share the base name, such that the clearinghouse can be an authority over the Aggregate Manager name. For example:

`urn:publicid:IDN+`**`geni.net:gpo`**`+authority+sa`

can be an authority over the Aggregate Manager

`urn:publicid:IDN+`**`geni.net:gpo:bbntest1`**`+authority+am`

But it could not be an authority over the Aggregate Manager

`Urn:publicid:IDN+`**`geni.net:bbntest1`**`+authority+am`

Because **`geni.net:gpo`** is not a prefix of **`geni.net:bbntest1.`**

10.4 Type

The <type> string indicates the type of the object being identified. Table 10-3 contains a list of type names in current use, along with a description of each. It is desirable that aggregate implementations use common names for GENI elements, but there is presently no registry for URN type names. Implementations should try to reuse existing names in use by other aggregates where possible.

Type name	Description
user	A GENI experimenter.
authority	A signing authority.
slice	A slice.
sliver	A sliver.

Table 10-3: GENI URN type names.

10.5 Name

The <name> string can be any valid transcribed public id. Use this field to ensure the entity (e.g., user, slice, or resource) is uniquely identified.

Note: For ProtoGENI compatibility, slice credentials must be signed by an authority with name "sa", e.g., `urn:publicid:IDN+gcf:gpo+authority+sa`.

11 Certificates

Certificates are used to authenticate actors in the GENI API.

The GENI Aggregate Manager API uses X.509 v3 identity certificates to bind public keys to identifiers (URNs). Only the holder of the private key that signed the certificate can act as the principal named by the URN.

In the GENI API, these certificates are used for both server side authentication and client side authentication in SSL connections (actually HTTPS). They are also used to identify the subject and target of credentials.

11.1 Format

A GENI certificate is an X.509 v3 [RFC 3280] certificate that specifies a GENI identifier (URN) in the X.509 v3 subjectAltName extension. The certificate is stored in PEM [RFC 1421] format. The GENI identifier (URN) is placed in URI format and begins with: 'URI:urn:publicid:IDN+'. The certificate's Common Name (CN) values for the Issuer and Subject are not specified by the GENI specifications and can be any valid common name. An example GENI certificate that uses a dotted notation for the common names can be seen in Appendix A.

11.2 Hierarchy

Certificate authority (CA) hierarchies are supported. In a CA hierarchy, a root CA can create normal certificates as well as intermediate CA certificates. Intermediate CAs are able to issue certificates that are verified by following the chain from the certificate to the intermediate CA's certificate to the root certificate. Typically, the verifier will only have the root CA's certificate installed for verification, and the intermediate CA's certificates is appended to the certificates it issues (called

PEM chaining). In the certificate listed in the appendix, the following lines declare that the subject is an intermediate CA:

```
X509v3 Basic Constraints: critical
    CA:TRUE
```

Since Aggregates and Clearinghouses are likely to only have the root CA's certificate installed, and not the intermediate CA certificates, all certificates signed by an intermediate CA should be chained. A chained certificate is simply a certificate that appends the issuer's certificate to the end of the file. For instance, if A is a root CA cert, B is an intermediate CA cert, and C is an end-user certificate, then C's chained certificate is:

```
C
B
A (optional)
```

An example chained certificate which shows the user cert, the intermediate CA cert, and the root CA cert (in that order from top to bottom) in chained PEM format can be seen in Appendix B.

12 Credentials

Credentials are used to authorize actions (where certificates authenticate and URNs identify). Credentials specify the privileges of an actor (the subject of the credential) on an object (the target of the credential). For instance, a slice credential gives an experimenter (the subject) privileges to allocate and remove resources from a slice (the target). The credential format that the GENI AM API uses is ProtoGENI's credential [12]. The GENI AM API adds some additional privileges (see Section 13) for PlanetLab compatibility.

Credentials can be delegated to other subjects. For instance a slice credential can be delegated from one experimenter to another. Since credentials assign privileges, this means that one user can assign some or all of their privileges to others.

GENI Credentials are signed XML containing:

- **Type** can be one of 'privilege', 'ticket', and 'capability'. The GENI AM API is only concerned with privilege credentials.
- **Serial** is a value specified by the issuer, and can be any string.
- **UUID** is unused.
- **Owner GID**, which is a PEM format X.509 certificate, containing the owner (entity the credential is being made for)'s URN in the Subject Alt Name field. See Section 10. Note that this may be a chained certificate.
- **Owner URN** to identify the owner (entity whose permissions are being specified). See Section 9.
- **Target GID**, an X.509 certificate identifying the target (e.g., a slice)
- **Target URN** naming the target. Note that the signer of the credential must either have the same URN as the target, or be an authority over the target URN’s namespace. See Section 9.
- **Expiration date** in ISO 8601 format when the credential becomes invalid (in or relative to UTC)
- List of **privileges**. See Section 12.
- **Signature** of the issuer of the credential. The issuer should be an authority over the Target's namespace. IE a ProtoGENI clearinghouse cannot issue credentials giving permissions over a PlanetLab slice.
- Credential **Format** per the schema described below.

- **Parent** If the credential is a delegated credential then the original credential is placed within its parent tag.

12.1 Signatures

The credential is signed using the XML Signature specification [13]. Implementations must be able to read and/or write credentials that conform to the XML Signature specification. The xmlsec software package [14] is one example of a software package that can be used to read and write compliant credentials. If a credential is delegated, then the owner creating the new (delegated) credential signs the new credential and the original signature and the new signature are placed in the <Signatures> section. An example credential appears below.

```
<?xml version="1.0"?>
<signed-credential>
  <credential xml:id="ref0">
    <type>privilege</type>
    <serial>8</serial>
    <owner_gid>certificate here</owner_gid>
    <owner_urn>urn:publicid:IDN+plc:gpo:site2+user+alice</owner_urn>
    <target_gid>certificate here</target_gid>
    <target_urn>urn:publicid:IDN+plc:gpo:site2+slice+alpha</target_urn>
    <uuid/>
    <expires>2012-07-14T19:52:08</expires>
    <privileges>
      <privilege>
        <name>resolve</name>
        <can_delegate>true</can_delegate>
      </privilege>
      <privilege>
        <name>embed</name>
        <can_delegate>true</can_delegate>
      </privilege>
    </privileges>
  </credential>
  <signatures>
    signature information here
  </signatures>
</signed-credential>
```

12.2 Credential Validation

Credentials must be internally valid, as well as valid for the operation being requested.

Internal credential validation requires:

- All of the signatures are valid and that the issuers trace back to trusted roots.
- The XML matches the credential schema.
- The issuer of the credential is the authority for the target's URN.

- All of the certificates presented in the credential are valid.
- The credential is not expired.
- If the credential has been delegated, also verify that:
  - The type of the child and parent are the same
  - Expiration of child is no later than parent
  - Expiration of child is no later than current time
  - The signature over the child credential is valid and signed by the subject of the parent credential
  - Any assigned privileges are a subset of parent privileges and the <can_delegate> bit is set to 1 in the parent for each delegated privilege.

Credentials are verified in the context of the operation being requested. A credential is verified as permitting an operation only if:

- The credential's subject matches the XML-RPC caller. IE the credential is giving permissions to the same entity making the call.
- The credential's target matches the specified target URN in the call. IE the credential is giving permissions over the entity that the caller is trying to act on.
- The privileges listed in the credential map to the operation permissions required to execute the desired function.
- The credential itself is valid (per above).

12.3 Implementation

Credentials can be signed by ‘xmlsec1’, a program distributed with the xmlsec library. This program will take an XML file that has a signature template appended to it and an xml:id attribute, and sign the portion of the XML document designated by the same xml:id using the provided key. The signature is placed within the appended signature template.

The signature template is the following (replace "ref0" with the xml:id of the XML section that is to be signed):

```
<Signature xml:id="Sig_ref0" xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-
c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
    <Reference URI="#ref0">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```
      <DigestValue></DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue />
    <KeyInfo>
      <X509Data>
        <X509SubjectName/>
        <X509IssuerSerial/>
        <X509Certificate/>
      </X509Data>
      <KeyValue />
    </KeyInfo>
</Signature>
```

For example, this is a command to sign an XML file with a signature appendage:

```
xmlsec1 sign --node-id "Sig_ref0" --privkey-pem alice.pkey,alice.cert
template.xml > signed.xml
```

This is a command to verify the signature:

```
xmlsec1 verify --node-id "Sig_ref0" --trusted-pem intermediate_ca_cert --
trusted-pem root_ca_cert signed.xml
```

13 Privileges

Most GENI API operations require the experimenter to have specific privileges. The set of privileges an experimenter has for a slice are encoded in the credential. The SFA outlines a candidate set of privileges and the rationale behind privileges. PlanetLab and ProtoGENI use overlapping terminology, but with slightly different semantics for the resulting operations. To achieve compatibility between the two control frameworks for this API, the minimal necessary changes were made.

Operation	Privileges
GetVersion	None
ListResources	*, authority, resolve
CreateSliver	*, sa, embed, control
DeleteSliver	*, sa, embed, control
SliverStatus	*, sa, embed, control
RenewSliver	*, sa, embed, control
Shutdown	*, sa

Table 13-1: GENI credential privileges

ProtoGENI uses a special privilege, "*" (an asterisk) to denote "all privileges". The GENI API adopted this special privilege to indicate that an experimenter can perform any GENI API operation.

The Table 13-1 lists the GENI API operations and the privileges (as embedded in credentials by the issuing clearinghouse) that allow an experimenter to perform each operation. An experimenter needs

only one of the privileges associated with each operation in order to perform that operation. Note that within PlanetLab and ProtoGENI, these privilege names may have additional specific meanings.

Further reconciliation or translation of privilege names remains a future task.

14 Future Plans

The GENI aggregate manager API is a first pass at GENI interoperability. It is expected to grow and change over time to accommodate new technologies, resources, and tools based on community contributions and feedback. Some potential modifications are outlined below:

- API Extensions
  - Multiple slivers per slice at a single aggregate.
  - UpdateSliver (modify the set of requested resources under the same slice name. May be a synonym for CreateSliver)
  - CreateSlivers (plural, a synonym for CreateSliver)
  - StartSliver (After allocating resources, boot or start them. CreateSliver currently does this. This operation would not be possible after Shutdown by users, only administrators.)
  - StopSliver (Currently done by DeleteSliver, this would stop a resource from running if that is meaningful, without de-allocating the resources.)
  - Tickets and delegating Privileges (GetTicket/RedeemTicket).
- Standardize 'touchpoint' formatting in RSpecs to support stitching.
- Shibboleth or other standard identity provider, as a database of users, and as an alternative to certificates for authentication
- Attribute based access control or other mechanisms for managing authorization and federation

15 Appendix A: Sample GENI API certificate

This certificate was printed from the original pem format using the following command:

```
$ openssl x509 -in cert.pem -text
```

Note that this user certificate contains the user's URN in the subjectAltName field, and the Issuer and Subject CN are related but unspecified dotted-notation identifiers.

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 3 (0x3)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: CN=plc.gpo.site2
        Validity
            Not Before: Jun 10 17:15:29 2010 GMT
            Not After : Jun  9 17:15:29 2015 GMT
        Subject: CN=plc.gpo.site2.jkarlin
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
```

```
                Modulus (2048 bit):
                    00:bd:94:7a:7c:b7:76:c9:58:24:15:5d:e9:bf:06:
                    12:63:d4:f2:47:c3:a0:4b:f0:06:eb:da:19:d6:7d:
                    81:07:d5:7f:64:ad:a3:aa:32:ce:32:6d:ed:54:ca:
                    a9:8e:61:9a:49:e8:db:a7:29:ff:7e:23:73:a5:fe:
                    45:79:f4:e7:1b:5f:34:7c:43:89:a1:a8:76:41:0e:
                    5a:66:e7:8f:28:9c:19:c0:54:21:fb:49:ca:60:d9:
                    20:f0:c9:85:58:d3:93:30:5f:36:bb:c9:3e:44:ee:
                    f0:3e:0f:4a:68:d2:77:33:48:2a:08:a7:e9:7c:41:
                    21:5a:68:26:9c:f0:b6:3a:76:42:78:d9:dd:32:92:
                    80:6c:4c:8c:fa:b9:45:38:2c:71:99:57:69:39:a3:
                    75:3d:65:b7:02:64:cf:3d:9c:1c:90:b6:fe:3b:38:
                    26:73:51:b7:6c:f7:0a:44:84:9c:35:58:88:78:3c:
                    f8:47:19:65:df:b6:4d:dc:69:07:09:d1:14:19:08:
                    14:a6:07:6e:19:de:5d:91:38:3b:7b:b8:4c:c9:a9:
                    e9:b1:d7:8c:80:b6:87:95:7c:28:3e:28:b9:73:43:
                    41:5c:55:ee:d0:d2:52:e1:cf:f3:f5:3e:7c:12:f7:
                    0e:20:ee:26:4a:28:e3:b5:8b:e3:84:7c:d4:4e:e4:
                    9a:31
                Exponent: 35 (0x23)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Alternative Name:
                URI:urn:publicid:IDN+plc:gpo:site2+user+jkarlin
    Signature Algorithm: md5WithRSAEncryption
        82:39:3f:b2:1b:85:7c:18:32:13:ea:6d:32:47:e6:a4:df:5d:
        4e:48:7e:95:96:41:3e:b7:71:9a:f9:9c:5b:7a:f1:34:04:ca:
        c7:21:26:31:4c:77:8c:b6:57:6e:02:32:8c:84:9f:cf:4b:3e:
        65:d4:97:76:56:fd:5c:05:5d:02:63:ca:e2:48:dd:54:07:60:
        35:8a:04:6c:52:5e:a5:ea:f9:66:16:54:e8:7c:32:89:a7:e8:
        46:5e:af:ea:3b:d6:29:0f:45:e3:80:46:53:d8:e2:bd:9a:68:
        2a:9e:52:72:6a:3b:2c:40:8a:79:6a:1f:df:34:ed:20:cc:c8:
        7f:2b
```

16 Appendix B: Chained Identity Certificate

This is a sample of a chained X509v3 identity certificate.

```
-----BEGIN CERTIFICATE-----
MIICpTCCAg4CAQMwDQYJKoZIhvcNAQEEBQAwGDEWMBQGA1UEAxMNcGxjLmdwby5z
aXRlMjAeFw0xMDA2MTAxNzE1MjlaFw0xNTA2MDkxNzE1MjlaMCAxHjAcBgNVBAMT
FXBsYy5ncG8uc2l0ZTIuamthcmxpbjCCASAwDQYJKoZIhvcNAQEBBQADggENADCC
AQgCggEBAL2Ueny3dslYJBVd6b8GEmPU8kfDoEvwBuvaGdZ9gQfVf2Sto6oyzjJt
7VTKqY5hmkno26cp/34jc6X+RXn05xtfNHxDiaGodkEOWmbnjyicGcBUIftJymDZ
IPDJhVjTkzBfNrvJPkTu8D4PSmjSdzNIKgin6XxBIVpoJpzwtjp2QnjZ3TKSgGxM
jPq5RTgscZlXaTmjdT1ltwJkzz2cHJC2/js4JnNRt2z3CkSEnDVYiHg8+EcZZd+2
TdxpBwnRFBkIFKYHbhneXZE4O3u4TMmp6bHXjIC2h5V8KD4ouXNDQVxV7tDSUuHP
8/U+fBL3DiDuJkoo47WL44R81E7kmjECASOjejB4MA8GA1UdEwEB/wQFMAMBAf8w
ZQYDVR0RBF4wXIYrdXJuOnB1YmxpY2lkOklETitwbGM6Z3BvOnNpdGUyK3VzZXIr
amthcmxpboYtdXJuOnV1aWQ6MDllM2I1ZTEtNzdjMy00OTRkLTk0YWYtZWQ3YjRj
YWY2YmJkMA0GCSqGSIb3DQEBBAUAA4GBAII5P7IbhXwYMhPqbTJH5qTfXU5IfpWW
QT63cZr5nFt68TQEyschJjFMd4y2V24CMoyEn89LPmXUl3ZW/VwFXQJjyuJI3VQH
YDWKBGxSXqXq+WYWVOh8Momn6EZer+o71ikPReOARlPY4r2aaCqeUnJqOyxAinlq
H9807SDMyH8r
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIICFTCCAX4CAQMwDQYJKoZIhvcNAQEEBQAwEjEQMA4GA1UEAxMHcGxjLmdwbzAe
Fw0xMDA2MTAxNzE1MjhaFw0xNTA2MDkxNzE1MjhaMBgxFjAUBgNVBAMTDXBsYy5n
cG8uc2l0ZTIwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAKfMXAqlXrR3+EfW
UOY2SCjbSd11+oKbj8RUkp3Axjnm02Wo5pOTrLSaFhORARcmtvsxyfNn6rEYBCJ0
T+oNAC5HwSRFBpWKiRtW43+iRO9RQaxFo6rsBem65AuZZC3V2jXMvPmI9DCmcibF
1v4rN3kTGw6WnC3joswqPnFcgolBAgMBAAGjejB4MA8GA1UdEwEB/wQFMAMBAf8w
ZQYDVR0RBF4wXIYrdXJuOnB1YmxpY2lkOklETitwbGM6Z3BvOnNpdGUyK2F1dGhv
cml0eStzYYYtdXJuOnV1aWQ6M2I1YjMyNjctY2MzZC00MmE1LTg3ZmEtYjJjMTY5
ODgyOWIzMA0GCSqGSIb3DQEBBAUAA4GBAGVnGyuPaQdvqr5sydIdxVcbG9Vo+RoN
weTaG8eU7oQNjeBp4IwgJkC++EKYudCcG6JIl2LiensB6mTYmkvf8GPIbTKDwCdj
UWKOoez+EiWNZl7PQDgq/wXKn54VctMuyJesFYaVoztIy8ngYIQRJPqsHQdE1suC
zgNeDVgGkGsz
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIB+DCCAWECAQMwDQYJKoZIhvcNAQEEBQAwEjEQMA4GA1UEAxMHcGxjLmdwbzAe
Fw0xMDA2MTAxNzE1MjhaFw0xNTA2MDkxNzE1MjhaMBIxEDAOBgNVBAMTB3BsYy5n
cG8wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJtvhZx43pjyYronJHqdoqxq
7ir8nxOtOHxWKnTLYCPGSK2W1AxUljeTbTu0QI22kzlqNnVHw6iigTS1jr9uVr0Z
ic5CtNPajt4kpcF6dFfIo7D+V10XJqy6uU++kkZ5qFt503KBMELm2pSiedrwIvxh
MEdErlAL99fAfsAGIMFZAgMBAAGjYzBhMF8GA1UdEQRYMFaGJXVybjpwdWJsaWNp
```

```
ZDpJRE4rcGxjOmdwbythdXRob3JpdHkrc2GGLXVybjp1dWlkOjNjY2NkNWM4LTEw
ODItNDU5OS04MTY4LTU1YTA5NjA3MjM4OTANBgkqhkiG9w0BAQQFAAOBgQBkufkv
HW3EooAEBz5LWnCCEZf0qR6o9cR9r8ZnkczoShgEPdEfnYBtQGE5a3kt5RXJvPKJ
iGsg/eWBYpUfsEcwFDYzIxoHNH/rmxgwy6mItIQ90dQNdVYLvXEhtrya+3dkVhPa
qhhEfubmtMeptqr40vuXaioWnBlY3CDRO88sew==
-----END CERTIFICATE-----
```

17 References

[1] Peterson, Larry. "Slice-Based Facility Architecture version 1.01." *www.cs.princeton.edu*. 8 Aug. 2008. Web.

[2] "PlanetLab." *www.planet-lab.org*.

[3] "ProtoGENI." *www.protogeni.net*.

[4] "Shibboleth." *http://shibboleth.internet2.edu/*.

[5] "XML-RPC." *www.xmlrpc.com*.

[6] "Python." *www.python.org*.

[7] "XML." *www.w3.org*.

[8] "RDF." *www.w3.org*.

[9] "Formal Public Identifiers." *www.wikipedia.org*.

[10] "URNs." *www.protogeni.net*. 25 May 2010.

[11] Viecco, Camilo. "Proposal: Use of URN's as GENI Identifiers (Version 0.3, Draft)." *gmoc.grnoc.iu.edu*.

[12] "Credentials." *www.protogeni.net*. 9 June 2010.

[13] "XML Signature Syntax and Processing (Second Edition)." *www.w3.org*. 10 June 2008

[14] "XML Security Library." *http://www.aleksey.com/xmlsec/*.