Internet-Draft
Common Authentication technology WG
<draft-ietf-cat-gssv2-javabind-spi-02.txt>

M. Smith
TIAA-CREF

October 1999

Expires: January 2000

A Service Provider API for GSS mechanisms in Java

1. Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

2. Abstract

This document specifies a "provider API" by which GSS mechanisms implemented in Java can be accessed through an intermediate "broker" or "shim" layer.

3. Acknowledgments

This document is the result of work done in the Common Authentication Technology (CAT) working group of the IETF. Special thanks are due to Mayank Upadhyay and Jack Kabat. Thanks also are due to John Linn for helpful comments on a preliminary version of this document. Dave Arnold's eagle eyes detected several errors and inconsistencies in an

earlier version. All errors and imbecilities, of course, remain the author's own.

4. Introduction

The Generic Security Service API (GSS-API)[1], a product of the CAT working group, specifies a language- and mechanism-independent interface by which application programs can use security services (e.g. authentication and privacy). Companion documents specify "language bindings" of the abstract GSS-API to particular languages.

For the Java language binding, it seems appropriate to define a type of GSS implementation consisting of a "shim" or "broker" layer, providing no security services itself. Actual security services would be provided by mechanism implementations (MIs). These MIs would register themselves with the "shim" layer, and be accessed by the shim layer through a "provider" API.

This document specifies the interface between "shim" layer implementations and MIs; means for registering the latter with the former; and means for controlling the "default mechanism" behavior of shims. The interface exposed by the shim layer to application programs consists of the interfaces defined for the Java GSS binding, as described in [2], plus the additional methods specified in the interface org.ietf.jgss-spi.MechanismManager, defined in this document (see section 10 below). Note that the interfaces of the Java-GSS binding remain the object of ongoing work; this document will be aligned with the final state of those interfaces. It is not expected that this alignment will involve changes to this document above the level of minor detail.

5. Components of this specification

This specification defines a java "package", to be named org.ietf.jgss-spi. This package contains the following interfaces:

```
org.ietf.jgss-spi.GSSMechanismManager extends
   org.ietf.jgss.GSSManager
org.ietf.jgss-spi.GSSMechanism
```

6. The shim layer and the mechanism implementation layer

A "shim" implementation consists of a class implementing the interface org.ietf.jgss-spi.GSSMechanismManager (which in turn extends the interface org.ietf.jgss.GSSManager defined in [2]).

7. Mechanism implementations and providers

A mechanism implementation (MI) consists of a class implementing the interface org.ietf.jgss-spi.GSSMechanism described in section 9.2 below. It is instances of classes implementing GSSMechanism which are "registered" by the "shim" as providers of security services.

Various means are provided for making MIs available to shim implementations. An array of names of classes implementing GSSMechanism may be listed in instances of java.security.Provider, indexed by a well-known property name (see 10 below). As part of its initialization, a shim implementation may query java.security.Security's getProviders() method and register any MIs found in installed java.security.Providers under this property name.

Providers containing this property name can also be passed to shim instances after their initialization, and any MIs which they contain added dynamically with instance scope (see 10.1.5).

Or the name of a single class may be listed in an instance of java.security.Provider, under a property name containing a dotted-decimal representation of an Oid. GSSMechanismManager provides methods which will permit such a Provider instance to be passed in and the GSSMechanisms it contains to be added dynamically with instance scope (see 10.1.4 below).

Alternatively, instances of classes implementing GSSMechanism can be passed directly (without being encapsulated in an instance of java.security.Provider) to GSSMechanismManager's insertMechImplementation() method, and inserted in the shim's list of implementations, with shim instance scope (see 10.1.3 below).

Besides providing methods to manage the repertoire of MIs, GSSMechanismManager also provides means to control the default behavior of shim instances (see sections 10.1.2 and 10.1.3 below).

8. Names, credentials, and contexts

MIs must implement the interfaces GSSName, GSSCredential, and GSSContext (defined in [2]). Note, however, that these interfaces incorporate some degree of multi-mechanism functionality; since MIs are not multi-mechanism, the functionality of some of the methods of these interfaces, as implemented by MIs, will be circumscribed. These limitations of functionality are described in detail in section 9.1 below.

Shim layers must also provide implementations of these three interfaces; each "shim" name or credential will serve as containers

for one or several MI names and credentials, and a shim context will serve as a container for precisely one MI context. What follows here is a general discussion of the relationship between (on the one hand) names, credentials, and contexts created by the shim and used by calling applications, and (on the other hand) those created by MIs. For more detail, see the discussion of MIs' implementations of individual methods of the three interfaces (below, sections 9.1.1, 9.1.2, and 9.1.3).

8.1. Names

The shim's version of createName() will produce a GSSName which includes mechanism-specific names for as many of its registered mechanisms as possible; a shim createName(), if no mechanism parameter is provided, will successively call the createName() method of each of its registered MIs. If an MI can generate a name corresponding to the parameters provided, its createName() method will return a GSSName (guaranteed, of course, to be a "name for mechanism") based on those parameters; otherwise, it will throw an exception, which will be caught internally by the shim's createName(). When the shim's createName() has finished going through its list of currently registered MIs, it will return an instance of a class implementing GSSName and containing all the GSSName instances successfully returned by the calls it made to the various MIs' createName() methods. If no mechanisms were registered or if none could successfully create a name from the parameters provided, the shim's createName() will throw an exception. (See 11.5 below for further considerations on this topic.)

8.2. Credentials

The shim's createCredential methods will in turn call the createCredential methods of one or more registered MIs. If the requesting application specified Oid(s) for specific mechanisms, only MIs implementing the requested mechanisms will be called; otherwise, the shim should call the createCredential methods of as many registered MIs as possible. If the requesting application provided a GSSName parameter, this must be a "container" name returned by the shim as decribed in the previous section. The shim must pass to each MI's createCredential call the MI-specific component of the shim's container name object that was created by that MI; if a given MI was unable to provide a component for this name, then it should not be called. Each MI so called will attempt to obtain and return an object implementing GSSCredential (with circumscribed, i.e. single-mechanism, functionality as described more fully in section 9.1 below.) The shim's credential implementation, as returned to the calling application, will act as a container for these various MI-

specific credentials. If no mechanisms were registered or if none could successfully create a credential from the parameters provided, the shim's createName() will throw an exception. (See 11.5 below for further considerations on this topic.)

The shim's implementation of the add() method of GSSCredential will call the createCredential method of the MIs (if any) registered for the Oid specified in the add() call, provided that the GSSName holder (if any) associated with this GSSCredential holder contains a component for that MI. If this process successfully returns a credential, that credential will be added to the set contained in the shim's GSSCredential implementation.

If there is no MI registered for this Oid, the shim will throw a GSSException with status BAD_MECH. If a GSSName holder was provided and it contained no GSSName element for any of the MIs implementing this mech, a GSSException will be thrown with status BAD_NAME. If MIs for this mech and appropriate GSSNames are found, but none of the MIs' createCredential() methods succeeds, the shim will throw an exception. (See 11.5 below for further considerations on this topic.)

8.3. Contexts

Contexts are created, in [2], by two versions of the createContext call: one which takes an interprocess token and reconstitutes a "freeze-dried" context, and others which take various optional parameters and are used to establish a new context ab ovo (either on the initiating or accepting side). Behavior of the first version (with the interprocess token) is implementation-dependent and will not be specified here. Behavior of the second version is, generally, as follows (see 9.2 below for more detail):

The shim's createContext call will return an object created by the shim and implementing GSSContext from [2]; this is referred to as the "context holder." Any parameters (names, credentials, etc.) supplied by the calling application will be stored away in this context holder, but no MIs will yet be called; an initiating application may still want to set various other context parameters, using the set() methods of GSSContext, which may affect the choice of a mechanism, and an accepting application has not yet provided an input context-establishment token.

When the requesting application finally calls the context holder's initSecContext() or acceptSecContext() methods, the shim will call one or more appropriate MI(s) in an attempt to see whether any of them can establish the context with the requested parameters. (If an Oid specifying a mechanism is provided, of course, only MIs implementing that mechanism will be called). As with the

createCredential call (above), if name or credential parameters are provided by the calling application, these must be the shim's container implementations of these interfaces, and the shim will provide, to each MI, only those name or credential components of the name or credential container that were originally provided by that MI.

9. The provider layer: details

Generally speaking, a mechanism implementation (MI) at the provider layer looks very much like a single-mechanism GSS implementation, with certain limitations and extensions. Such an implementation must implement the interface org.ietf.jgss-spi.GSSMechanism described below (section 9.2), and the interfaces

GSSName
GSSCredential
GSSSecurityContext

from [2].

Note, however, that these three GSS interfaces are designed, in general, to encapsulate multiple mechanisms, and the interfaces at the provider layer are designed only to encapsulate a single mechanism. Thus, some versions of the methods defined in these three interfaces are superfluous for MIs. These differences are described in the following subsections.

For the sake of brevity, and to avoid duplication, the semantics of methods in the MI version of these interfaces will be described only to the extent that they differ from those defined in [2]. Methods which do not differ as to signature, functionality, or exception generation will not be re-described here.

(For some second thoughts on this subject, see 11.4 below).

9.1. Functional limitations in MIs

9.1.1. GSSName

MI implementations of GSSName can only be "names-for-mechanism" (MNs) as defined in [1].

9.1.1.1. canonicalize()

The canonicalize() method in MI implementations of GSSName will throw a GSSException with major status of BAD_MECH if the Oid parameter is

not the one for the mechanism supported by this MI.

9.1.2. GSSCredential

9.1.2.1. add()

The method add() in MIs throws a GSSException with major status of BAD_MECH, since MIs are not multi-mechanism.

9.1.2.2. getName(GSSOIDString mechoid)

This version of the getName method, which takes an OID designating the mechanism, will throw a GSSException with major status of BAD_MECH unless the Oid is the one for the mechanism supported by this MI.

9.1.2.3. getUsage(GSSOIDString mechoid)

This version of the getUsage method which takes an OID designating the mechanism wil throw a GSSException with major status of BAD_MECH unless the Oid is the one for the mechanism supported by this MI.

9.1.2.4. getRemainingAcceptLifetime()

This method will throw a GSSException with major status of BAD_MECH unless the Oid parameter is the one for the mechanism supported by this MI.

9.1.2.5. getRemainingInitLifetime()

This method will throw a GSSException with major status of BAD_MECH unless the Oid parameter is the one for the mechanism supported by this MI.

9.1.3. GSSContext

MIs handle context initiation off the getContextForInit() method of GSSMechanism (q.v., 9.2 below). This method takes parameters representing the services being requested of this context, and therefore, MI implementations of GSSContext no-op the various pre-initiation methods of GSSContext, sc.

```
requestMutualAuth
requestReplayDet
requestSequenceDet
requestCredDeleg
requestAnonymity
```

```
requestConf
requestInteg
requestLifetime
setChannelBinding
```

9.2. The interface GSSMechanism

For MIs, the interface GSSMechanism in effect replaces the interface GSSManager defined in [2], and it has a certain family resemblance to that interface, since it constitutes the factory class that MIs use to create credentials, names, and contexts. Since an MI is single-mechanism, however, and because this interface is not intended to be used by application programmers, this interface can be somewhat stripped-down as compared with GSSManager, for performance reasons inter alia.

9.2.1. Constants

GSSMechanism defines the following bit-mappings of a short integer field, to be used in indicating the services requested for a context:

```
public static short GSSServiceDelegReq = 1 << 0; // Delegation
public static short GSSServiceMutualReq = 1 << 1; // Mutual
    authentication
public static short GSSServiceReplayDetReq = 1 << 2; // Replay
    detection
public static short GSSServiceSequenceReq = 1 << 3; // Sequence
    enforcement
public static short GSSServiceAnonReq = 1 << 4; // Anonymity
public static short GSSServiceConfReq = 1 << 5; // Confidentiality
public static short GSSServiceDelegReq = 1 << 6; // Integrity
```

9.2.2. Methods

9.2.2.1. No-parameter constructor

Classes that implement GSSProviderImplementation must provide a no-parameter constructor.

9.2.2.2. acceptable()

The full signature of this method is:

```
public boolean acceptable
  (Object token,
  GSSCredential cred,
  Object[] channelBindings) throws GSSException;
```

This method indicates whether the Object passed as 'token' is usable by the implementing mechanism as a context-establishment token, with the credential in the second parameter and the channel bindings in the third. The "cred" parameter may be null; this value requests default credential behavior. If provided, 'cred' must be a GSSCredential object returned by this MI; if not, a GSSException will be thrown with major status of DEFECTIVE_CREDENTIAL. The channelBindings parameter may be null, in which case the context will not be bound to a channel.

This method is intended to be used by the acceptor-side version of the createContext() method of the shim's GSSManager implementation to determine which of the registered mechanism implementations to use when a context-establishment token is submitted and a context is to be initially created for acceptance.

9.2.2.3. getContextForAccept()

The full signature of this method is:

```
public GSSContext getContextForAccept
    (Object token,
    GSSProviderCredential cred,
    Object[] channelBindings)
      throws GSSException;
```

This method creates a security context using the MI's mechanism, using the "token" parameter as a context-establishment token, with the credential in the second parameter and the channel bindings in the third. 'cred' may be null; this requests default behavior. If non-null, 'cred' must be a GSSCredential object returned by this MI. The channelBindings parameter may be null, in which case the context will not be bound to a channel.

This method is intended to be used by the acceptor-side version of the createContext() method of the shim's GSSManager implementation when a context is to be initially created for acceptance.

9.2.2.4. getContextForInit()

The full signature of this method is:

```
public GSSContext getContextForInit
```

```
      (GSSCredential cred,
       GSSName targname,
       int lifetimeReq,
       Object[] channelBindings,
       short servicesRequested)
           throws GSSException;
```

This method requests the provider to create and return a GSSContext object for the mechanism this MI implements, suitable for use on the initiating side of the context. The "cred" parameter may be null; this value requests default credential behavior. If non-null, "cred" must be a GSSCredential object returned by this MI's createCredential() method. "targname" is a GSSName obtained from this MI's createName() method designating the intended acceptor of the context. "lifetimeReq" is the requested lifetime of the context (see the relevant section of [1]); zero requests a mechanism-specific default. "channelBindings" may be null and if so, the context will not be bound to a channel. "servicesRequested" is a bit-mapped field whose bits have the meanings described in 9.2.1 above.

This method is intended to be used by the initiator-side version of createContext() in the shim layer.

9.2.2.5. **initable()**

The full signature of this method is:

```
   public boolean initable
       (GSSCredential cred,
       GSSName targname,
       int lifetimeReq,
       Object[] channelBindings,
       short servicesRequested);
```

This method requests the provider to determine whether a GSSContext object can be created for the mechanism this provider implements, suitable for use on the initiating side of the context, with the parameters supplied. The "cred" parameter may be null; this requests default credential behavior. "targname" is a GSSName obtained from this MI's createName() designating the intended acceptor of the context. "lifetimeReq" is the requested lifetime of the context (see the relevant section of [1]); zero requests a mechanism-specific default. "channelBindings" may be null and if so, the context will not be bound to a channel. "servicesRequested" is a bit-mapped field whose bits have the meanings described in 9.2.1 above.

This method is intended to be used by the initiating side's version of the createContext() method of GSSManager in the shim layer, in

determining which of the mechanisms available to use when initiation of a security context is requested.

9.2.2.6. createCredential

The full signature of this method is:

```
public GSSCredential createCredential
    (GSSName aName,
    int lifetimeReq,
    int usage)
      throws GSSException;
```

This method is functionally equivalent to the method of the same name in GSSManager (see [2], section 6.1.13), except that it does not permit an Oid parameter to be provided (since MIs are single-mechanism). 'aName' may be null, in which case default principal credentials are being requested; otherwise, 'aName' must be a GSSName returned by the createName() method of this MI.

9.2.2.7. GSSName createName(byte[] externalrep, Oid nameSpace)

This method is functionally identical to the method of the same name in GSSManager ([2], 6.1.7), except that the 'name' object returned is an MI-specific "name for mechanism" (MN).

9.2.2.8. Oid[] getNames()

Similar to getNamesForMech of GSSManager([2], 6.1.5) but takes no Oid parameter identifying the mechanism (since MIs are single-mechanism).

9.2.2.9. Oid[] getMech()

Returns an Oid identifying the mechanism implemented by this MI.

9.2.2.10. short getServices()

This method returns a bit-mapped short integer indicating to services available from this MI (see 9.2.1 above).

10. The shim's management API

The interface org.ietf.jgss-spi.GSSMechanismManager (which in turn extends the interface org.ietf.jgss.GSSManager defined in [2]) defines both the application-related services of GSS itself, derived from GSSmanager, and an additional set of management functions which permit the repertoire of MIs available to be changed, and default behavior to be specified. These management functions are the subject of this section.

Conceptually, the shim maintains an ordered list of GSSMechanism implementation instances (MIIs) known to it, and a corresponding ordered list of Oids implemented by these MIIs; that is, a given MII, and the Oid of the mechanism it implements, occur at the same ordinal position on their respective lists. A shim's getMechs() method must return Oids in the order in which they occur on this internal list.

Default behavior depends, in part, on the ordering of the internal MII list. Versions of GSSManager methods (createName, createCredential, createContext) which do not specify a mechanism Oid will cause the list of installed MIIs to be "visited" in the order defined by this internal list. Context creation, in particular, will use the first MII on the list for which the following conditions are satisfied:

1) Credential and/or name elements exist, provided by this MII, in any instances of the shim's container credential or name classes provided as parameters to the context-creation call; and

2) The MII can satisfy the services (e.g. anonymity, mutual authentication, privacy, etc.) requested in the context creation call.

Various means exist for adding MIIs to the list:

1) An array of names of classes implementing GSSMechanism may be listed in instances of java.security.Provider, indexed by the property name "GSSMechanism". As part of its initialization, a shim implementation may query java.security.Security's getProviders() method, call the constructor for each class listed under this property name, and add each GSSMechanism instance so obtained to its list of MIIs. The shim's MII list will then (after initialization) hold all the GSSMechanism implementations listed in installed system-wide Providers, in the order in which those providers were installed.

2) After initalization, GSSMechanismManager's addProvider(java.security.Provider prov) method may be called; the 'prov' parameter will be searched for a property named

"GSSMechanism," and MIIs installed from the class list associated with that property, as described in the previous paragraph. This method adds MIIs from providers to a specific instance of the shim (not system-wide). Shim implementors may check with the local SecurityManager or AccessController if it is considered necessary to restrict this operation to privileged code.

3) The name of a single class may be listed in an instance of java.security.Provider, under a property name of the form "GSSMechanism-x.y.z...", where 'x.y.z...' is the dotted-decimal representation of the Oid of the mechanism implemented by that class. GSSMechanismManager's addProvider(Oid mech, Provider prov) method will call the constructor for a class whose name is found in such a property, and add the resulting GSSMechanism instance to the MII list. This version of addProvider, like the previous one, operates on a particular instance of the shim, rather than system-wide. Shim implementors may check with the local SecurityManager or AccessController if it is considered necessary to restrict this operation to privileged code.

4) Finally, entries may be inserted directly into the list, or removed from it, by the methods removeMechImplementation() and insertMechImplementation(), defined below (section 10.1.1. and 10.1.2). These methods operate on a particular shim instance, not system-wide, and additionally offer means to change the ordering of MIIs on the internal list. Shim implementors may check with the local SecurityManager or AccessController if it is considered necessary to restrict these operations to privileged code.

All the methods discussed above, of course, affect the list of Oids as well as the list of MIIs.

10.1. Methods of the management API

10.1.1. GSSMechanism[] getMechImplementations();

This method returns the list of the GSSMechanism instances registered in this instance of the shim, in the same order as the corresponding Oids would be returned by the getMechs() method defined in GSSmanager. This order will correspond to the order in which the shim will call its registered GSSMechanisms when it handles a method to which a default mechanism parameter has been provided.

10.1.2. GSSMechanism removeMechImplementation(int position);

This method removes the MII at position 'position' from this instance of the shim's internal MII list, and returns a reference to it. This

method will throw an array index out of bounds exception if 'position' is out of bounds.

10.1.3. void insertMechImplementation (GSSMechanism mii, int position);

This method will insert 'mii' at 'position' in the internal list of MIIs maintained by this instance of the shim, and will move the former occupant of 'position' and all its successors one step down the list. The Oid of the mechanism implemented by 'mii' will be inserted in the corrresponding position in this instance's Oid list (i.e. the list of Oids that would be returned by getMechs()). This method will not throw an index out of bounds exception; if 'position' is past the end of the current list, this method will add 'mii' at the end. If 'position' is negative, 'mii' will be added at the beginning.

10.1.4. void addProvider(Oid mech, java.security.Provider prov) throws GSSException;

This method permits MIs "contained" in java.security.Provider objects to be added dynamically to shim instances.

The Provider 'prov' will be queried for a property with the name "GSSMechanism-x.y.z...", where "x.y.z..." is the dotted-decimal representation of the Oid in 'mech'. The value of this property should be the name of a class implementing GSSMechanism. An instance of the class will be created and added at the end of this shim instance's current list of MIIs.

This version of addprovider() will throw a GSSException with major status of BAD_MECH if no property with the appropriate name was found, a GSSException with status UNAVAILABLE if the class denoted by the property value could not be loaded, and a GSSException with status FAILURE if the class denoted by the property value was available but did not implement GSSMechanism.

10.1.5. boolean addProvider(java.security.Provider prov)

The Provider 'prov' will be queried for a property with the name "GSSMechanism". The value of this property should be an array of names of classes implementing GSSMechanism. An instance of each class will be created and added at the end of this shim instance's current list of MIIs. No exceptions will be thrown by this version of addProvider(); it will return 'true' if any MIIs were successfully added, 'false' otherwise.

11. Topics for further discussion

11.1. SPNEGO

One reader of an early version of this document has raised the question whether SPNEGO should be regarded as "just another mechanism provider," or be included in the shim implementation. Either approach is possible with the interfaces defined in this document. In favor of including SPNEGO in the shim are arguments from performance and interoperability. On the opposite side of the question is the desire to minimize the size and complexity of the shim. The author sees no clear-cut case for either approach; comments are sought from interested parties.

11.2. A concrete class?

This document specifies interfaces only and leaves open the possibility of multiple "shim" implementations. Some participants in the Working Group have expressed the view that either in this document or in the Java-GSS bindings document [2], a concrete class ought to be specified which would have shim functionality and consitute the "vanilla" or "default" or "reference" implementation of GSS for Java.

11.3. SecurityManager/AccessController

Shim implementors may wish to use the authorization services of the Java environment to control access to the management functions defined in section 10.1 above. The Working Group may wish to consider whether the means by which such checks are made should be standardized, and if so, how: e.g. do we prefer to use some existing type of Permission object and standardize its parameters for use by shims, or define a new Permission object type, as part of this specification?

11.4. New interfaces, or function limitation?

This document at present specifies that MIs implement the standard GSS-API interfaces (defined in [2]) for name, credential, and context; but by virtue of the inherently single-mechanism nature of MIs, this document specifies certain limitations of function in these interfaces as implemented by MIs. Perhaps it might be preferable to define separate interfaces for MIs, modeled on the standard GSS interfaces but with inherently single-mechanism semantics and suitably adjusted parameters and exception repertoires.

11.5. Comprehensive failure

It will sometimes occur that none of the registered MIs will be able to establish a context or create a name or credential successfully; but they may all fail for different reasons. What exception(s) should the shim report in this case?

11.6. Diagnostic information from initable() and acceptable()

Do the methods initable() and acceptable() in GSSMechanism (sections 9.2.2.2 and 9.2.2.5) need to return more detailed diagnostic information?

12. Security Considerations

This entire document deals with security.

13. Conclusion

This document specifies a "provider API" by which GSS mechanisms implemented in Java can be accessed through an intermediate "broker" or "shim" layer, and an API by which the repertoire of mechanisms and default behavior can be managed.

14. References

[1] J. Linn, "Generic Security Service Application Program Interface, Version 2, Update 1," Internet-Draft, <draft-ietf-cat-rfc2078bis-08.txt>, December 1998

[2] Jack Kabat, "Generic Security Service API Version 2 : Java bindings," Internet-Draft, <draft-ietf-cat-gssv2-javabind-02.txt>, August 1998

15. Author's Address

Michael Smith
TIAA-CREF
730 Third Avenue
Mailstop 485-27-02
New York, NY 10017

USA

Phone: 212 490 9000 x 1760
Email: ms@gf.org