

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: September 19, 2013

Wilkinson  
YFS  
March 18, 2013

Integrating rxgk with AFS  
draft-wilkinson-afs3-rxgk-afs-02

## Abstract

This document describes how the new GSSAPI-based rxgk security class for RX is integrated with the AFS application protocol. It describes a number of extensions to the basic rxgk protocol, clarifies a number of implementation issues, and provides values for the application-specific elements of rxgk.

## Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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."

This Internet-Draft will expire on September 19, 2013.

## Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Internet-Draft

Integrating rxgk with AFS

March 2013

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction . . . . .</a>	<a href="#">3</a>
<a href="#">1.1.</a>	<a href="#">The AFS-3 Distributed File System . . . . .</a>	<a href="#">3</a>
<a href="#">1.2.</a>	<a href="#">rxgk and AFS . . . . .</a>	<a href="#">3</a>
<a href="#">1.3.</a>	<a href="#">Requirements Language . . . . .</a>	<a href="#">4</a>
<a href="#">2.</a>	<a href="#">Security Index . . . . .</a>	<a href="#">4</a>
<a href="#">3.</a>	<a href="#">Key Negotiation . . . . .</a>	<a href="#">4</a>
<a href="#">3.1.</a>	<a href="#">The AFSCombineTokens Operation . . . . .</a>	<a href="#">4</a>
<a href="#">4.</a>	<a href="#">Tokens . . . . .</a>	<a href="#">6</a>
<a href="#">4.1.</a>	<a href="#">Container . . . . .</a>	<a href="#">6</a>
<a href="#">4.2.</a>	<a href="#">Token Encryption . . . . .</a>	<a href="#">6</a>
<a href="#">4.3.</a>	<a href="#">Token Contents . . . . .</a>	<a href="#">7</a>
<a href="#">5.</a>	<a href="#">Authenticator Data . . . . .</a>	<a href="#">8</a>
<a href="#">6.</a>	<a href="#">Client Tokens . . . . .</a>	<a href="#">8</a>
<a href="#">6.1.</a>	<a href="#">Keyed Clients . . . . .</a>	<a href="#">8</a>
<a href="#">6.2.</a>	<a href="#">Unkeyed Clients . . . . .</a>	<a href="#">9</a>
<a href="#">7.</a>	<a href="#">Server to Server Communication . . . . .</a>	<a href="#">9</a>
<a href="#">7.1.</a>	<a href="#">Token Printing . . . . .</a>	<a href="#">9</a>
<a href="#">8.</a>	<a href="#">Declaring rxgk Support for a Fileserver . . . . .</a>	<a href="#">9</a>
<a href="#">9.</a>	<a href="#">Per Server Keys . . . . .</a>	<a href="#">10</a>
<a href="#">10.</a>	<a href="#">Securing the Callback Channel . . . . .</a>	<a href="#">12</a>
<a href="#">11.</a>	<a href="#">IANA Considerations . . . . .</a>	<a href="#">13</a>
<a href="#">12.</a>	<a href="#">AFS-3 Registry Considerations . . . . .</a>	<a href="#">13</a>
<a href="#">13.</a>	<a href="#">Security Considerations . . . . .</a>	<a href="#">13</a>
<a href="#">13.1.</a>	<a href="#">Downgrade attacks . . . . .</a>	<a href="#">13</a>
<a href="#">13.2.</a>	<a href="#">Per Server Keys . . . . .</a>	<a href="#">13</a>
<a href="#">13.3.</a>	<a href="#">Combined Key Materials . . . . .</a>	<a href="#">13</a>
<a href="#">14.</a>	<a href="#">References . . . . .</a>	<a href="#">14</a>
<a href="#">14.1.</a>	<a href="#">Informational References . . . . .</a>	<a href="#">14</a>
<a href="#">14.2.</a>	<a href="#">Normative References . . . . .</a>	<a href="#">14</a>
<a href="#">Appendix A.</a>	<a href="#">Acknowledgements . . . . .</a>	<a href="#">14</a>
<a href="#">Appendix B.</a>	<a href="#">Changes . . . . .</a>	<a href="#">15</a>
<a href="#">B.1.</a>	<a href="#">Since 00 . . . . .</a>	<a href="#">15</a>
<a href="#">B.2.</a>	<a href="#">Since 01 . . . . .</a>	<a href="#">15</a>
	<a href="#">Author's Address . . . . .</a>	<a href="#">15</a>

## 1. Introduction

rxgk [[I-D.wilkinson-afs3-rxgk](#)] is a new GSSAPI-based [[RFC2743](#)] security layer for the RX [[RX](#)] remote procedure call system. The rxgk specification details how it may be used with a generic RX application. This document provides additional detail specific to integrating rxgk with the AFS-3 distributed file system.

### 1.1. The AFS-3 Distributed File System

AFS-3 is a global distributed network file system. The system is split into a number of cells, with a cell being the administrative boundary. Typically an organisation will have one (or more) cells, but a cell will not span organisations. Each cell contains a number of file servers which contain collections of files ("volumes") which they make available to clients using the AFS-3 protocol. Clients access these files using a service known as the cache manager.

In order to determine which server a particular file is located upon, the cache manager looks up the location in the volume location database (vlldb) by contacting the vlserver. Each cell has one or more vl servers, which are synchronised using an out-of-band mechanism.

### 1.2. rxgk and AFS

This document describes special integration steps needed to use rxgk with AFS-3 database servers (the PR and VL rx services) and file servers (the RXAFS, RXAFSCB, and AFSVol rx services), as well as specifying application-specific portions of the rxgk specification for use by these services. Other AFS-3 services are not covered by this document; the generic rxgk document applies to them. AFS-3 differs from the standard rxgk implementation in that it does not require GSSAPI negotiation with each server. Instead, a client performs GSSAPI negotiation just once (with the vlserver), receiving a token usable with any server in the cell that has the cell-wide

key. Traditional AFS rxkad authentication required that the cell-wide key be distributed to all servers in the cell, both database servers and file servers. rxgk can operate in such a fashion, with the cell-wide key shared amongst all servers.

For more complex cell topologies, rxgk also supports configurations where (some) file servers do not have the cell-wide key, by means of an extended version of the CombineTokens RPC. This new RPC, AFSCombineTokens, takes a server identifier, and will return a token encrypted with a key for a specific server. AFSCombineTokens also provides support for indicating whether a specific server is rxgk capable, allowing cells to securely migrate to rxgk from other

security mechanisms.

We also define mechanisms for securing the callback channel that is created between fileserver and client.

### [1.3.](#) Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

## [2.](#) Security Index

When used within the AFS protocol, rxgk has an RX securityIndex value of 4.

## [3.](#) Key Negotiation

An AFS cell wishing to support rxgk MUST run an rxgk key negotiation service, as specified in [[I-D.wilkinson-afs3-rxgk](#)], on each of its vlserver. The service MUST listen on the same port as the vlserver.

The GSS identity `afs-rxgk@_afs.<cellname>` of nametype GSS\_C\_NT\_HOSTBASED\_SERVICE is the acceptor identity for this service. Where multiple vlserver exist for a single cell, all of these servers must have access to the key material for this identity, which MUST be identical across the cell. Clients MAY use the presence of

this identity as an indicator of rxgk support for a particular cell. Clients that wish to support cells using other rx security objects MAY downgrade if this identity is not available.

Tokens returned from the GSSNegotiate call MUST only be used with database servers. Tokens for file servers MUST be obtained by calling AFSCombineTokens before each server is contacted.

### [3.1.](#) The AFSCombineTokens Operation

AFS extends the existing CombineTokens operation to provide a general token manipulation service. This operation takes a user token, an optional cache manager token, options for enctype and security level negotiation with the server, and a destination identifier. It returns a token specific to the specified destination, and a structure containing some cleartext information describing the returned token.

```
AFSCombineTokens(IN RXGK_Data *token0,  
                 IN RXGK_Data *token1,  
                 IN RXGK_CombineOptions *options,  
                 IN afsUUID destination,  
                 OUT RXGK_Data *new_token,  
                 OUT RXGK_TokenInfo *token_info) = 3;
```

token0: An rxgk token obtained using the GSSNegotiate RPC.

token1: Either an rxgk token obtained using the GSSNegotiate RPC, or empty (zero-length).

options: An RXGK\_CombineOptions structure containing a list of enctypes acceptable to the client and a list of security levels acceptable to the client.

destination: The UUID of the server new\_token is intended for. Fileserver UUIDs may be obtained from the VLDB in the same call that returns their addresses.

new\_token: The output rxgk token, or empty (zero-length).

token\_info: Information describing the returned token.

The AFSCombineTokens call MUST only be performed over a secured rxgk connection. AFSCombineTokens MUST NOT be offered over an RXGK\_LEVEL\_CLEAR connection. Servers MUST reject all attempts to perform this operation over channels that do not offer integrity protection.

Clients which are caching the results of RPCs on behalf of multiple users (such as a traditional AFS Cache Manager), SHOULD provide both the user's token (as token0) and a token generated from an identity that is private to the cache manager (as token1). This prevents a user from poisoning the cache for other users. Recommendations on keying cache managers are contained in [Section 6.1](#).

Clients which are working on behalf of a single user can provide an empty token1, but MUST use AFSCombineTokens to obtain a destination specific token for each fileserver they contact.

Clients using a printed token (see [Section 7.1](#)) MUST provide that token as token0. token1 MUST be empty. Printed tokens cannot be combined with any other token, and servers MUST reject attempts to do so.

If the server is unable to perform the AFSCombineTokens operation with the given arguments, a nonzero value is returned in the

errorcode field of token\_info; errorcode is zero for a successful AFSCombineTokens operation. If errorcode is nonzero, the values of the other fields in token\_info and the value of new\_token are undefined.

If the returned token is zero-length, then the destination does not support rxgk, and the client MAY fall back to using a different authentication mechanism for that server. An rxgk capable client operating within an rxgk enabled cell MUST NOT downgrade its choice of security layer in any other situation.

Other aspects of the operation of AFSCombineTokens, including the values for the errorcode field of token\_info and the combination of keys and tokens, are the same as the CombineTokens RPC, documented in [CombineTokens call](#), documented in [\[I-D.wilkinson-afs3-rxgk\]](#).

## [4. Tokens](#)

### [4.1. Container](#)

rxgk tokens for AFS take the form of some key management data, followed by an encrypted data blob. The key management data (a version number, followed by an [\[RFC3961\]](#) encryption type) allows the server receiving a token to identify which pre-shared key has been used to encrypt the core token data.

```
struct RXGK-TokenContainer {
    afs_int32 kvno;
    afs_int32 enctype;
    opaque    encrypted_token<>;
};
```

The RXGK-TokenContainer structure is XDR encoded and transported within the 'token' field of the RXGK\_ClientInfo structure specified in [\[I-D.wilkinson-afs3-rxgk\]](#).

### [4.2. Token Encryption](#)

Token contents are encrypted using a pre-shared key. rxgk supports both the use of a single cell-wide key and the use of per-server keys. The cell-wide key must be installed on all servers which are capable of accepting cell-wide tokens. Cell-wide keys should be for a selected [RFC3961](#) encryption mechanism that is supported by all servers within the cell that will accept cell-wide tokens. Per-server keys should be for an encryption mechanism that is supported by both the destination server and the negotiation service. The management of per-server keys is discussed in more detail in

### [Section 13.2.](#)

Key rollover is permitted by means of a key version number. When the key is changed, a different key version number MUST be selected. Servers SHOULD accept tokens using old keys until the lifetime of all existing tokens has elapsed.

Encryption is performed over the XDR encoded RXGK-Token structure,

using the [RFC3961](#) encrypt operation, with a key usage value of RXGK\_SERVER\_ENC\_TOKEN (defined in [[I-D.wilkinson-afs3-rxgk](#)]). The encrypted data is stored in the `encrypted_token` field of the `TokenContainer` structure described in [Section 4.1](#).

### [4.3](#). Token Contents

The token itself contains the information expressed by the following RPC-L:

```
struct RXGK_Token {
    afs_int32 enctype;
    opaque K0<>;
    RXGK_Level level;
    rxgkTime start_time;
    afs_int32 lifetime;
    afs_int32 bytelife;
    rxgkTime expirationtime;
    struct PrAuthName identities<>;
};
```

`enctype`: The [RFC3961](#) encryption type of the session key contained within this ticket.

`K0`: The session key (see [[I-D.wilkinson-afs3-rxgk](#)] for details of how this key is negotiated between client and negotiation service).

`level`: The security level, as defined in [[I-D.wilkinson-afs3-rxgk](#)], that MUST be used for this connection.

`start_time`: The time at which the token's validity begins. Servers MUST reject attempts to use tokens with a `start_time` value later than the current time.

`lifetime`: The maximum number of seconds that a key derived from `K0` may be used for, before the connection is rekeyed. If 0, keys have no time-based limit.

`bytelife`: The maximum amount of data (expressed as the log base 2 of



the number of bytes) that may be transferred using a key derived from K0 before the connection is rekeyed. If 0, there is no data-based limit on key usage.

**expirationtime:** The time (expressed as an rxgkTime) beyond which this token may no longer be used. Servers MUST reject attempts to use connections secured with this token after this time. A time of 0 indicates that this token never expires.

**identities:** A list of identities represented by this token. struct PrAuthName is the identity structure defined in [\[I-D.brashear-afs3-pts-extended-names\]](#).

## [5.](#) Authenticator Data

The appdata opaque within the RXGK\_Authenticator structure contains the results of XDR encoding the RXGK\_Authenticator\_AFSAppData structure. The uuid field contains the UUID of the client.

```
struct RXGK_Authenticator_AFSAppData {  
    afsUUID uuid;  
};
```

## [6.](#) Client Tokens

In order to protect users of a multi-user cache manager from each other, it must be impossible for an individual user to determine the key used to protect operations which affect the cache. This requires that the cache manager have key material of its own which can be combined with that of the user. This functionality is provided by the AFSCombineTokens call specified earlier in this document. However, this call requires that a cache manager have access to a token for this purpose.

### [6.1.](#) Keyed Clients

When a host already has key material for a GSSAPI mechanism supported by rxgk, that material MAY be used to key the client. The client simply calls the rxgk negotiation service using the relevant material, and obtains a token. The client SHOULD frequently regenerate this token, to avoid combined tokens having unnecessarily close expiration times. The client SHOULD NOT regenerate this token so often so as to place excessive load on the vlservers.

It is recommended that identities created specifically for use by a

cache manager have the name `afs3-callback@<hostname>` where `<hostname>` is the fully qualified domain name of the cache manager.

## [6.2.](#) Unkeyed Clients

When a client has no key material, it is possible that an anonymous GSSAPI connection may succeed. Clients MAY attempt to negotiate such a connection by calling `GSS_Init_sec_context()` with the `anon_req_flag` [[RFC2743](#)] and the default credentials set.

## [7.](#) Server to Server Communication

A number of portions of the AFS protocol require that servers communicate amongst themselves. To secure this with rxgk we require both a mechanism of generating tokens for these servers to use, and a definition of which identities are permitted for authorisation purposes. We refer to the process of forging tokens for local use, given access to the cell-wide pre-shared key, as "token printing".

### [7.1.](#) Token Printing

A server with access to the cell-wide pre-shared key may print its own tokens for server to server access. To do so, it should construct a token with suitable values. The list of identities in such a token MUST be empty. It can then encrypt this token using the pre-shared key, and use it in the same way as a normal rxgk token. The receiving server can identify it is a printed token by the empty identity list.

The session key within a printed token MUST use the same encryption type as the pre-shared key. When connecting to a fileserver, a client SHOULD use the `AFSCombineTokens` service as discussed above to ensure that they are using the correct key for the fileserver.

## [8.](#) Declaring rxgk Support for a Fileserver

The `AFSCombineTokens` call has specific behaviour when a destination endpoint does not support rxgk. Implementing this behaviour requires that the vlserver be aware of whether a fileserver supports rxgk.

Fileservers currently register with the vlserver using the `VL_RegisterAddrs` RPC. Fileservers which support rxgk MUST call this RPC over a rxgk protected connection. The vlserver then infers rxgk support from the rx security layer used in registration. To prevent

downgrade attacks, once a fileserver has registered as being rxgk capable, the vlserver MUST NOT remove that registration without

administrator intervention.

Once a fileserver has been marked as supporting rxgk, VL\_RegisterAddrs calls for that fileserver MUST only be accepted over an rxgk protected connection. vlservers MUST only accept calls to VL\_RegisterAddrs from a printed token, an administrator, or the identity registered for the fileserver using a prior call to VL\_RegisterAddrsAndKey.

## [9.](#) Per Server Keys

The provisioning of servers with their own keys, rather than the cell-wide master key, requires the ability to maintain a directory of these keys on the vlserver, so that the AFSCombineTokens RPC can encrypt the outgoing token with the correct key. The manner in which this directory is maintained is left to the implementor, who MAY decide to use a manual, or out of band, key management system. Otherwise, the automated keying mechanism described as follows will be used.

Implementations supporting automatic key management through the AFS-3 protocol MUST provide the VL\_RegisterAddrsAndKey RPC (similar to the VL\_RegisterAddrs RPC). This RPC is called by a fileserver to register itself with the VLDB; it MUST be called over an rxgk-secured connection. For the purpose of this RPC, the fileserver acts as the client and the vlserver as the server. Once the RPC completes, the client can generate a key to be used as the fileserver's server key.

vlservers MUST NOT permit calls to VL\_RegisterAddrsAndKey for UUIDs which already exist within the vlserver, unless that UUID already has a server-specific key registered. When a new fileserver first registers with the vlserver using VL\_RegisterAddrsAndKey, the vlserver MUST store the identity used to make this connection. The vlserver MUST only permit subsequent calls to VL\_RegisterAddrsAndKey for this UUID when they come from this identity, an administrator, or a printed token. New fileserver UUIDs register themselves with the vlserver in a "leap of faith", binding a GSSAPI identity to the fileserver UUID for future authenticated operations. Fileservers

SHOULD use VL\_RegisterAddrsAndKey to rekey themselves periodically, in accordance with key lifetime best practices.

The VL\_RegisterAddrsAndKey RPC is described by the following RPC-L:

Wilkinson

Expires September 19, 2013

[Page 10]

---

Internet-Draft

Integrating rxgk with AFS

March 2013

```
struct RXGK_ServerKeyDataRequest {
    afs_int32 enctype<>;
    opaque nonce1[20];
};

struct RXGK_ServerKeyDataResponse {
    afs_int32 enctype;
    afs_int32 kvno;
    opaque nonce2[20];
};

typedef opaque keyDataRequest<>;
typedef opaque keyDataResponse<>;
VL_RegisterAddrsAndKey(
    IN afsUUID *uuidp,
    IN afs_int32 spare1,
    IN bulkaddrs *ipaddr,
    IN afs_int32 secIndex,
    IN keyDataRequest *request,
    OUT keyDataResponse *response) = XXX;
```

uuidp: The fileserver's UUID.

spare1: Unused. (Clients SHOULD pass zero.)

ipaddr: The list of addresses to register as belonging to this fileserver.

secIndex: The index of the security mechanism for which a key is being set. For rxgk, this value MUST be 4.

keyDataRequest: An opaque blob of data, specific to the security

mechanism defined by secIndex. For rxgk, it is the XDR-encoded representation of an RXGK\_ServerKeyDataRequest structure.

keyDataResponse: An opaque blob of data, specific to the security mechanism defined by secIndex. For rxgk, it is the XDR-encoded representation of an RXGK\_ServerDataResponse structure.

The client provides, in the RXGK\_ServerKeyDataRequest structure, a list of the [RFC3961](#) encryption types that it will accept as a server key. It also provides a nonce containing 20 random data bytes.

The server selects an encryption type shared by it and the client, and returns that, along with 20 bytes of random data that it has generated, in RXGK\_ServerKeyDataResponse. If there is no common encryption type, then the server MUST fail the request.

The server key can then be derived by both client and server using

```
random-to-key(PRF+(K0, K, nonce1 || nonce2));
```

random-to-key is the function specified by the [RFC3961](#) profile of the encryption type chosen by the server and returned in enctype.

PRF+ is the function of that name specified by [[RFC4402](#)].

K0 is the master key of the current rxgk session, as originally determined by the GSSNegotiate call.

K is the key generation seed length as specified in enctype's [RFC3961](#) profile.

|| is the concatenation operation.

## [10](#). Securing the Callback Channel

AFS has traditionally had an unprotected callback channel. However, extended callbacks [[I-D.benjamin-extendedcallbackinfo](#)] require a mechanism for ensuring that callback breaks and, critically, data updates, are protected. This requires that there is a strong connection between the key material used initially to perform the

RPC, and that which is used to protect any resulting callback. We achieve this using the cache manager token discussed in [Section 6.1](#), which is required in order for a client to accept secure callbacks.

A cache manager may set a key for secure callbacks by issuing the following RPC (in the RXAFS service):

```
RXAFS_SetCallbackKey(afs_int32 securityIndex,  
                     opaque mech_data<>) = XXX;
```

**securityIndex:** The securityIndex of the mechanism for which this key is being set. In the rxgk case, this will be rxgk's security index, 4.

**mech\_data:** This contains the security object specific data. In rxgk's case this is an XDR encoded RXGK-Token structure.

When used with rxgk, this RPC MUST be performed over an rxgk protected connection established using solely the cache manager's token. This connection MUST have a security level of RXGK\_LEVEL\_CRYPT.

If a fileserver receives an RXAFS\_SetCallbackKey call protected with

a different cache manager identity than the previous call from that client, it MUST break all secure callbacks held by that client using the old key before this RPC completes.

Only RPCs issued over an rxgk protected connection should receive rxgk protected callbacks.

The fileserver MUST only send rxgk protected callbacks when one of the identities performing the RPC establishing that callback matches the identity associated with that clients callback channel.

## [11.](#) IANA Considerations

This memo includes no request to IANA.

## [12.](#) AFS-3 Registry Considerations

This document requests that the AFS-3 registry allocate code points for the new RPCs AFSCombineTokens (for the RXGK service), RegisterAddrsAndKey (for the VL service), and SetCallbackKey (for the RXAFS service).

## [13.](#) Security Considerations

### [13.1.](#) Downgrade attacks

Using the presence of a GSSAPI key to determine a cell's ability to perform rxgk is vulnerable to a downgrade attack, as an attacker may forge error responses. Cells which no longer support rxkad should remove their afs@REALM and afs/cell@REALM Kerberos keys.

### [13.2.](#) Per Server Keys

The mechanism for automatically registering per-server keys is potentially vulnerable, as it trades a short-lived key (the rxgk session key, which protects the key exchange) for a long-lived one (the server key). There is precedent for this sort of key exchange, such as when using kadmin to extract a new kerberos keytab.

### [13.3.](#) Combined Key Materials

As described in [Section 6](#), combined tokens are used to prevent cache poisoning attacks on multi-user systems. In order for this protection to be effective, cache managers MUST NOT provide user access to keys produced through the combine tokens operation, unless

Wilkinson	Expires September 19, 2013	[Page 13]
-----------	----------------------------	-----------

---

Internet-Draft	Integrating rxgk with AFS	March 2013
----------------	---------------------------	------------

those keys will not be used by the cache manger itself.

## [14.](#) References

### [14.1.](#) Informational References

[RX] Zeldovich, N., "RX protocol specification".

[I-D.benjamin-extendedcallbackinfo]  
Benjamin, M., "AFS Callback Extensions (Draft 14)",

[draft-benjamin-extendedcallbackinfo-02](#) (work in progress),  
December 2011.

## [14.2.](#) Normative References

- [I-D.brashear-afs3-pts-extended-names]  
Brashear, D., "Authentication Name Mapping extension for  
AFS-3 Protection Service",  
[draft-brashear-afs3-pts-extended-names-09](#) (work in  
progress), March 2011.
- [I-D.wilkinson-afs3-rxgk]  
Wilkinson, S., "rxgk: GSSAPI based security class for RX",  
[draft-wilkinson-afs3-rxgk-00](#) (work in progress),  
January 2010.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2743] Linn, J., "Generic Security Service Application Program  
Interface Version 2, Update 1", [RFC 2743](#), January 2000.
- [RFC3961] Raeburn, K., "Encryption and Checksum Specifications for  
Kerberos 5", [RFC 3961](#), February 2005.
- [RFC4402] Williams, N., "A Pseudo-Random Function (PRF) for the  
Kerberos V Generic Security Service Application Program  
Interface (GSS-API) Mechanism", [RFC 4402](#), February 2006.
- [RFC4506] Eisler, M., "XDR: External Data Representation Standard",  
STD 67, [RFC 4506](#), May 2006.

## [Appendix A.](#) Acknowledgements

rxgk has been the work of many contributors over the years. A  
partial list is contained in the [[I-D.wilkinson-afs3-rxgk](#)]. All

Wilkinson	Expires September 19, 2013	[Page 14]
-----------	----------------------------	-----------

---

Internet-Draft	Integrating rxgk with AFS	March 2013
----------------	---------------------------	------------

errors and omissions are, however, mine.

## [Appendix B.](#) Changes



[B.1.](#) Since 00

Add references to RX and XDR specifications.

Add introductory material on AFS.

Change expirationTime to be expressed using the rxgkTime type.

Document how encryption types are chosen for printed tokens, and how they are used against file servers.

Expand security considerations section to cover combined tokens.

Rename AFS\_SetCallbackKey as RXAFS\_SetCallbackKey.

[B.2.](#) Since 01

Rename RXAFS\_SetCallbackKey to RXAFS\_SetCallBackKey.

Add an AFS-3 Registry Considerations section.

Clarify the vlserver/dbserver/fileserver relationship.

AFSCombineTokens prototype changes.

Clarify the scope of the document.

Use a leap of faith for RegisterAddrsAndKey.

Specify the nametype of the acceptor identity.

Author's Address

Simon Wilkinson  
Your File System Inc

Email: [simon@sxw.org.uk](mailto:simon@sxw.org.uk)