Network Working Group S. Wilkinson
Internet-Draft YFS
Intended status: Informational B. Kaduk
Expires: November 22, 2015 MIT
May 21, 2015

Integrating rxgk with AFS
draft-wilkinson-afs3-rxgk-afs-08

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.

Table of Contents

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, but leaves some aspects of the protocol as application-specific. This document resolves the application-specific portions of rxgk for use with the AFS-3 distributed file system, and provides additional detail specific to integrating rxgk with AFS-3.

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 fileservers 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 (vldb) by contacting the vlserver. Each cell has one or more vlservers, which are synchronised using an out-of-band mechanism.

User and group information is stored in the protection database (prdb), which is made available by the ptserver(s), colocated with the vlservers. Fileservers check with the prdb before granting access to files which are subject to access control.

1.2. rxgk and AFS

This document describes the 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 a standard rxgk deployment 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 database server in the cell, as described in Section 5. Traditional AFS rxkad authentication required that the

cell-wide key be distributed to all servers in the cell, both database servers and file servers, making no distinction between tokens used for database servers and file servers. rxgk can operate in a similar fashion, with a cell-wide key shared amongst all servers, but is not limited to doing so.

For more complex cell topologies, rxgk also supports configurations where (some) file servers do not have the cell-wide key. Tokens encrypted in these server-specific keys are returned by an extended version of the CombineTokens RPC, AFSCombineTokens. AFSCombineTokens also provides a mechanism for indicating whether a specific server is rxgk capable, allowing cells to securely migrate to rxgk from other security mechanisms.

1.3. Providing Keys for the Callback Channel

The AFS-3 protocol provides a mechanism by which a client can obtain a promise from a fileserver to "call back" when a particular piece of data is changed, so that the client does not need to check with the fileserver for the current-ness of the data every time it is used. At present, this takes the form of a single bit of information about whether the callback is still valid, with no authentication of the callback break. It is desired that future work expand the callback channel to convey more than a single bit of information, and provide an authenticated (and potentially confidential) channel for updating callback promises.

This document provides a mechanism to establish a key and token that can be used to provide a secure callback channel. Though the format of that token is flexible and not specified in this document, this document does need to specify a mechanism by which a callback key can be established between the two parties. This is done by means of the authenticator's appdata field, binding a callback key to an rx connection, so that all callbacks generated by that connection will use the indicated callback key.

1.4. 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-3 protocol, rxgk has an RX securityIndex value of 4.

3. Authenticator Data

The appdata opaque within the RXGK_Authenticator structure used in the rx challenge/response authentication exchange contains the results of XDR [RFC4506] encoding the RXGK_Authenticator_AFSAppData structure.

```
struct RXGK_Authenticator_AFSAppData {
    afsUUID client_uuid;
    RXGK_Data cb_tok;
    RXGK_Data cb_key;
    afs_int32 enctype;
    afsUUID target_uuid;
};
```

client_uuid the UUID of the client.

cb_tok the rxgk token to be used for secure callbacks created by RPCs over this connection. In some implementations this token may be empty (zero-length).

cb_key the raw key material (k0) to which cb_tok corresponds, to be used as the master key for the secure callback connections created by RPCs over this connection.

enctype the [RFC3961] enctype of the cb_key key material.

target_uuid the UUID of the server being authenticated to (if applicable). Database servers do not have UUIDs; when authenticating to database servers, this field should be set to all zero bits. File server UUIDs may be obtained from the VLDB in the same call that returns their addresses.

4. Application-Specific Constant

The constant RXGK_MAXDATA takes the value 1048576 for use with AFS-3.

5. 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 vlservers. 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 vlservers 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. Note that not all GSS mechanisms can expose to the initiator whether or not a given acceptor identity exists.

5.1. Application-Specific GSSNegotiate Behavior for AFS-3

The input and output opaques of the GSSNegotiate RPC are left as implementation-defined, as needed by the implementation to retain information across subsequent calls during a single GSS negotiation loop.

5.2. Token Applicability

Tokens returned from the GSSNegotiate and CombineTokens calls MUST only be used with database servers. Tokens for fileservers MUST be obtained by calling AFSCombineTokens (Section 9) before each server is contacted.

rxgk tokens are in general only usable with the particular rx service that produced them. For the AFS-3 protocol, the database server services are grouped together to accept a common type of token, and the file server services are grouped together to accept a different common type of token, but it is important to emphasize that a token for a database server will not in general be useful against a file server, and vice versa. Tokens for database servers are obtained from the standard rxgk negotiation services, but tokens for file servers are obtained through a new procedure, the AFSCombineTokens RPC.

6. Token Format

This section defines the format of rxgk tokens for use with the AFS-3 protocol. The same layout is used for database server tokens and file server tokens, but file server tokens may be encrypted in a different key than database server tokens.

6.1. 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 RFC 3961 encryption type) allows the server receiving a token to identify which key has been used to encrypt the core token data.

```
struct RXGK_TokenContainer {
    afs_uint32 kvno;
    afs_int32 enctype;
    opaque    encrypted_token<>;
};
```

The RXGK_TokenContainer structure is XDR encoded and transported wherever a token is used, such as in the 'token' field of the RXGK_ClientInfo structure specified in [I-D.wilkinson-afs3-rxgk].

6.2. Token Encryption

rxgk supports encrypting tokens with either a single cell-wide key or with per-file-server keys. The cell-wide key must be installed on all database servers in the cell, and may additionally be installed on non-database file servers when per-file-server keys are not in use. Cell-wide keys should be for a selected RFC 3961 encryption mechanism that is supported by all servers within the cell that will use that key. Per-server keys should be for an encryption mechanism that is supported by both the destination server and the negotiation service on the vlserver. The management of per-server keys is discussed in more detail in Section 14.2.

Key rollover is permitted by means of a key version number. When a key is changed, whether cell-wide or per-server, a different (larger) key version number MUST be selected. Servers SHOULD accept tokens using old keys until the lifetime of all existing non-printed (see Section 10.1) tokens has elapsed. Services using printed tokens should be prepared to regenerate those tokens in the case of key rollover.

Encryption is performed over the XDR encoded RXGK_Token structure, using the RFC 3961 encrypt operation, with a key usage value of RXGK_SERVER_ENC_TOKEN (defined in [I-D.wilkinson-afs3-rxgk]). The enrypted data is stored in the encrypted_token field of the RXGK_TokenContainer structure described in Section 6.1.

6.3. 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;
    afs_uint32 lifetime;
    afs_uint32 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.

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 value of 0 indicates that this token never expires. It is RECOMMENDED that an expirationtime of 0 is only used for printed tokens.

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

7. Cache Manager Tokens

Some deployment scenarios for AFS-3 involve multi-user machines with a single Cache Manager that fetches data on the users' behalf. When multiple users have access to the same content, data that is fetched on the behalf of one user may be cached and re-displayed to a second user, without re-fetching it from the fileserver hosting the data. The initial data aquisition is authenticated by the first user's

credentials, and if only that user's credentials are used, it may be possible for a malicious user or users to "poison" the cache for other users, and introduce bogus data.

In order to protect users of a multi-user cache manager from each other, it is possible to give the cache manager its own token, which can be combined (Section 9) with the users' tokens so that the user may be authenticated at the fileserver while still preserving the integrity of the data obtained by the cache manager. In order to obtain a token, the cache manager must have some means of acquiring/ using key material.

7.1. Keyed Clients

When a host already has key material for a GSSAPI mechanism supported by the vlserver, that material MAY be used to key the cache manager. The cache manager simply calls the rxgk negotiation service using the relevant material, and obtains a token. This token is a database server token; there is no need in the protocol for it to be usable as the user_tok input to AFSCombineTokens or for there to be an entry in the protection database corresponding to the cache manager's GSS identity. The cache manager should frequently regenerate its token, to avoid combined tokens having expiration times which are substantially earlier than the expiration time of the corresponding user credentials. The cache manager should not regenerate this token so often so as to place excessive load on the vlservers.

It is recommended that GSS identities created specifically for use by a cache manager have the name afs3-callback@<hostname> of name type GSS_C_NT_HOSTBASED_SERVICE where <hostname> is the fully qualified domain name of the machine upon which the cache manager is running.

7.2. 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.

In some cases a cache manager may not have any dedicated credentials, but have user credentials from multiple different users. These tokens could be combined using the RXGK_CombineTokens operation and the combined token used as a proxy cache manager token. However, conspiring malicious users could still be able to manipulate the cache, and the differing token expiration times for user tokens would make cache management quite complicated with this approach. As such, it is not recommended for general use.

8. Combining Tokens

This section describes the server-side behavior of the RXGK_CombineTokens operation for the AFS-3 protocol.

There are no application-specific fields in RXGK_Token, so only the behavior for combination of identity information remains to be specified.

The identity lists in the 'identities' fields of the two tokens are combined via order-preserving concatenation and placed in the 'identities' field of the output token.

Printed tokens (Section 10.1) cannot be combined with any other token, and servers MUST reject attempts to do so, whether via CombineTokens, AFSCombineTokens, or any other token-combining procedure. AFSCombineTokens with a printed user_tok and an empty cm_tok is not considered to be token combination for this purpose.

9. The AFSCombineTokens Operation

AFS extends the existing CombineTokens operation to provide a more featured token manipulation and conversion 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 file server identifier. It returns a token specific to the specified destination fileserver, and a structure containing some information describing the returned token. AFSCombineTokens is the only way to obtain a valid file server token (other than printing a token, see Section 10.1).

```
AFSCombineTokens(IN RXGK_Data *user_tok,
                 IN RXGK_Data *cm_tok,
                 IN RXGK_CombineOptions *options,
                 IN afsUUID *destination,
                 OUT RXGK_Data *new_token,
                 OUT RXGK_TokenInfo *token_info) = TBD;
```

user_tok: An rxgk token for the vlserver.

cm_tok: Either an rxgk token for the vlserver, 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. File server 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. This integrity guarantee protects the returned token information (token_info) as well as the options and destination arguments submitted to the server.

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 user_tok) and a token generated from an identity that is private to the cache manager (as cm_tok). This prevents a user from poisoning the cache for other users. Recommendations on keying cache managers are contained in Section 7.1.

The output token from AFSCombineTokens is a token specific to the fileserver indicated by the destination argument. As such, it is not a valid input token for a successor AFSCombineTokens operation, as the input tokens for AFSCombineTokens must be tokens for the vlserver. To prevent key-reuse attacks, the token master key in the output token must be unique per destination file server; the destination UUID is incorporated into the key derivation procedure to ensure this property.

Clients using a printed token (see Section 10.1) MUST provide that token as user_tok. cm_tok MUST be empty.

The server uses a zero-length new_token to indicate that the generation of rxgk tokens for the specified fileserver cannot work at the present time. Upon receipt of such a zero-length new_token, 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. (Such a client may still not attempt to use rxgk at all for an AFS cell if it has determined that there is no suitable GSS acceptor identity to be used for that cell.)

In other cases where the server is unable to perform the AFSCombineTokens operation with the given arguments, a nonzero value

is returned. Clients MUST NOT use such an error as an indication to fall back to to a different security class.

The 'identities' list from user_tok is copied to the 'identities' field of the new_token. The 'identities' list from cm_tok is discarded unused.

Other aspects of the operation of AFSCombineTokens, including the combination of keys and tokens, are largely the same as the CombineTokens RPC, documented in [I-D.wilkinson-afs3-rxgk] and Section 8. However, the AFSCombineTokens operation needs to include the destination file server's UUID in the key combination process to ensure that the resulting key is unique for each file server (and different from the key in the input tokens); AFSCombineTokens must also handle the case where the supplied cm_tok is absent (empty). In the two-token case, the KRB-FX-CF2 operation is still used, but the pepper1 and pepper 2 inputs will both include the destination UUID:

```
pepper1 :=  "AFS" || 00 || destination || enctype
pepper2 := "rxgk" || 00 || destination || enctype
```

where the strings "AFS" and "rxgk" exclude the NUL terminator; 00 is a NUL octet; destination is the XDR-encoding of the destination afsUUID; enctype is the enctype selected by the server and returned in the enctype field of token_info, encoded as a 32-bit integer in network byte order; and || is the concatenation operator. In the one-token case,

```
Kn := random-to-key(PRF+(K0, pepper0))
pepper0 := "rxgkAFS" || 00 || destination || enctype
```

where the string "rxgkAFS" excludes the NUL terminator. Note that the PRF+ function here is the one used in the KRB-FX-CF2 operation specified in [RFC6113], which differs from the PRF+ function specified in [RFC4402] and used elsewhere in this document. random-to-key is the function specified by the RFC3961 profile of the selected enctype.

10. Server to Server Communication

A number of portions of the AFS-3 protocol require that servers communicate amongst themselves. To name a limited subset of examples, file servers must register their location (IP addresses) with the vldb, and must query the prdb when serving data; moving volumes from one file server to another requires that the file servers communicate with each other directly.

A server with the cell-wide shared key can forge a token for its use in server-to-server communication, which we refer to as "token printing". Printed tokens take on a special form (Section 10.1) and are limited in that they cannot be combined with any other token.

However, file servers with a server-specific key (that is, without the cell-wide shared key), can only print a token to themselves. Such tokens are not usable to communicate with database servers or other file servers. As such, file servers with a per-server key will need GSS credentials (but, as with keyed clients, not necessarily entries in the protection database) in order to function. These credentials can be used to acquire an rxgk token, allowing queries to the database servers. They can also be used to register the file server in the vldb, and to create and update the file server's server-specific key in the vldb.

10.1. 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 database server 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, place it in an RXGK_TokenContainer describing the key used to perform the encryption, and use it in the same way as a normal rxgk token. The receiving server can identify it as a printed token by the empty identity list.

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

File servers with per-server keys may also print tokens, though these tokens are in general of limited utility. (Being file server tokens, they are not valid inputs to AFSCombineTokens, etc..)

10.2. 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 vldb have a record of whether a fileserver supports rxgk.

Fileservers currently register with the vlserver using the VL_RegisterAddrs RPC. This document introduces an extended version, VL_RegisterAddrsAndKey (Section 10.3), and either one may be used to indicate that a fileserver supports rxgk. Fileservers which support rxgk MUST call these RPCs over an 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 and VL_RegisterAddrsAndKey from a printed token, an administrator, or the identity registered for the fileserver using a prior call to VL_RegisterAddrsandKey.

There are two tracks for registering a file server as being rxgk-enabled; one for file servers with the cell-wide key, and another for file servers with per-server keys.

10.2.1. File Servers With the Cell-Wide Key

When a file server that will use the cell-wide key is registered as rxgk-capable, there is no need to register a new key for that server (and in fact it would be actively harmful!), so there is no need to use VL_RegisterAddrsAndKey. In this case, VL_RegisterAddrs is sufficient, and using a printed token for the rxgk connection for VL_RegisterAddrs indicates that the file server possesses the cell-wide key. Since the file server has the cell-wide shared key, it will get its key updated when the cell-wide key is updated, and does not need to update its own key separately. As such, it will never need to call VL_RegisterAddrsAndKey.

10.2.2. File Servers With Per-Server Keys

This section describes the case when the automated keying mechanism described in Section 10.3 is used. If the record of per-server keys in the vldb is being manually maintained, cell administrators should manually register the file servers in the vldb using VL_RegisterAddrs instead.

Since the goal is to establish a per-server key, VL_RegisterAddrsAndKey is necessary for the first call. However, best practices require that the file server change its long-term key periodically, so it must retain the ability to perform subsequent VL_RegisterAddrsAndKey calls in the future, to register those new keys in the vldb. For this reason, a printed token is not a useful choice for performing the initial call to VL_RegisterAddrsAndKey, since only a printed token would be able to perform a subsequent call. The printed token would require the cell-wide shared key, eliminating any benefit from having a server-specific key. As such, a regular (non-printed) token is required for the initial call to

VL_RegisterAddrsAndKey. A cell administrator's token could be used, but it is advantageous to allow file servers with per-server keys to operate without intervention by the central cell administrators (so that these file servers could be run solely by a local administrator without need for central administrator intervention).

Thus, it is expected that a file server with a per-server key will have a dedicated GSS identity and credentials that it will use for registering with the vldb (VL_RegisterAddrsAndKey) and that will also be used for securing the file server's regular connections to the database servers during normal operation. The vlserver will store in the vldb what GSS identity is used to perform VL_RegisterAddrsAndKey for a given file server UUID, and allow that identity to perform successor calls to VL_RegisterAddrsAndKey and VL_RegisterAddrs for that UUID.

Is is RECOMMENDED that GSS identities created solely for use on file servers with per-server keys be of the form afs3-fileserver@<hostname> of name type GSS_C_NT_HOSTBASED_SERVICE.

10.3. Registering Per Server Keys

The provisioning of file servers with their own keys, rather than the cell-wide master key, requires the ability to maintain a directory of these keys in the vldb, 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, 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 a secure connection that provides confidentiality protection.

For the purpose of this RPC, the fileserver acts as the client and the vlserver as the server. Once the RPC completes, both peers of the RPC call can generate a key to be used as the fileserver's long-term server key.

vlservers SHOULD NOT permit calls to VL_RegisterAddrsAndKey for fileserver UUIDs which already exist within the vldb, unless that UUID already has a server-specific key registered. Requiring the separation facilitates a workflow wherein existing servers retain the cell-wide key, and new file servers are created with per-server keys. Data volumes can then be gradually migrated to the new file servers, and old file servers decommissioned. Permitting file servers to

convert from cell-wide key to per-server keys would involve complicated access checking and update logic for which it is difficult to ensure correctness of implementation.

The VL_RegisterAddrsAndKey RPC is described by the following RPC-L:

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

struct RXGK_ServerKeyDataResponse {
    afs_int32 enctype;
    afs_uint32 kvno;
    opaque nonce2[20];
};

const RXGK_MAXKEYDATAREQUEST = 16384;
const RXGK_MAXKEYDATARESPONSE = 16384;
typedef opaque keyDataRequest<RXGK_MAXKEYDATAREQUEST>;
typedef opaque keyDataResponse<RXGK_MAXKEYDATARESPONSE>;
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.

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 kvno field of the RXGK_ServerKeyDataResponse is used to indicate to the client what key version number it should use for the key it will compute using these nonces. The kvno will be used in the RXGK_TokenContainer bearing file server tokens for this file server, to indicate which key was used to encrypt the RXGK_Token.

The vlserver MUST store the identity list from the token used to make this connection. The vlserver MUST only permit subsequent calls to VL_RegisterAddrsAndKey for this UUID when they come over a connection authenticated with that same identity list, an administrator's token, or a printed token. Such subsequent calls using an administrator's token or a printed token do not update the identity list associated with this UUID's key. New fileserver UUIDs register themselves with the vldb 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.

For rxgk, the file server key can then be derived by both client and server using

```
    random-to-key(PRF+(K0, K,
                       pepper || 00 || nonce1 || nonce2 || enctype));
```

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].

[[The PRF+ function defined in RFC 4402 specifies that the values of the counter 'n' should begin at 1, for T1, T2, ... Tn. However, implementations of that PRF+ function for the gss_pseudo_random() implementation for the krb5 mechanism have disregarded that specification and started the counter 'n' from 0. Since there is no interoperability concern between krb5 gss_pseudo_random() and rxgk key derivation, implementations of the RFC 4402 PRF+ function for rxgk key derivation should use the RFC 4402 version as specified, that is, with the counter 'n' beginning at 1.]]

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

K is the key generation seed length as specified in enctype's RFC3961 profile.

pepper is the ASCII string "RXGKRegisterAddrsAndKey" (without trailing NUL).

00 is a NUL octet.

enctype is the selected enctype, encoded as a 32-bit integer in network byte order.

|| is the concatenation operation.

11. 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 by binding the key used to secure the callback connection into the authenticator used to create the original rxgk connection. Callbacks created as a result of RPCs performed on that rxgk connection will use the callback key given in the authenticator.

11.1. Lifetime and scope of the callback channel

The RXGK_Authenticator_AFSAppData structure contains a key and enctype, but no key version number field. This restricts the connection to only ever having one key to secure callbacks created as a result of calls on that connection, even if there are multiple Rx challenge/response exchanges where a new authenticator could be constructed. This is acceptable, because if the client needs to rotate the key used for secure callbacks to it, the client can initiate a new connection to the server, with a new callback key.

It may be reasonable for a cache manager to only ever use one key for secure callbacks (until the cache manager is restarted), such as in a cell where all fileservers have the cell-wide shared key or where all fileservers are equally trusted. Alternately, a cache manager may use just one callback key per fileserver. In either case, which key to use for incoming callback connections is known just from the context of the connection, so there is no need to provide a callback token in the authenticator.

In all cases, both cache manager and file server must retain the callback key until all callbacks using that key are expired.

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

12. IANA Considerations

This memo includes no request to IANA.

13. AFS-3 Registry Considerations

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

14. Security Considerations

14.1. 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.

14.2. 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.

14.3. Combined Key Materials

As described in Section 7, 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 those keys will not be used by the cache manger itself.

15. References

15.1. Informational References

[RX]   Zeldovich, N., "RX protocol specification", October 2002.

[I-D.benjamin-extendedcallbackinfo]
Benjamin, M., "AFS Callback Extensions (Draft 14)", draft-benjamin-extendedcallbackinfo-02 (work in progress), December 2011.

15.2. 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.

[RFC6113] Hartman, S. and L. Zhu, "A Generalized Framework for Kerberos Pre-Authentication", RFC 6113, April 2011.

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 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 fileservers.

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.

B.3. Since 02

Deal with fallout of errorcode's removal from RXGK_TokenInfo.

Rework "securing the callback channel".

B.4. Since 03

Clarify the distinction between dbserver and fileserver tokens.

AFSCombineTokens is the only way to get file server tokens.

Add new kind of PrAuthName, PRAUTHTYPE_EMPTY.

Specify how cache manager token identities are stored in file server tokens.

Place bounds on some XDR opaque arrays.

Expound more about printed tokens, for dbservers and fileservers.

B.5. Since 04

Rearrange content within the document in attempt to give a more coherent structure and improve readability.

Add specifications for the remaining pieces of rxgk behavior which the core document left as application-specific.

Change the token format. Instead of having the last entry in the identities list be the CM identity, use an explicit separate field for the identity to be used for callbacks.

As a result, PRAUTHTYPE_EMPTY is no longer necessary.

General edits for grammar and readability.

Add security considerations for the DoS attach that is possible by setting fake callback keys.

Add a clarifying note for the RFC 4402 PRF+ implementation.

B.6. Since 05

Remove start_time from the token format.

Remove the SetCallBackKey RPC, in favor of putting a callback key in the authenticator appdata. This provides a simpler solution to the problem of establishing a secure callback channel.

While here, add the server UUID into the appdata as well as the client UUID, to prevent some possible routes to data corruption.

B.7. Since 06

General edits for clarity.

Use afs_uint32 for token lifetimes, to match the core spec.

B.8. Since 07

Incorporate the destination UUID and target enctype into AFSCombineTokens key generation.

Add (fixed) pepper strings for AFSCombineTokens and RegisterAddrsAndKey key generation.

General editing for clarity.

Use unsigned types for kvnos.

Use a pointer type for afsUUID RPC arguments.

Authors' Addresses

Simon Wilkinson
Your File System Inc

Email: simon@sxw.org.uk

Benjamin Kaduk
MIT Kerberos Consortium

Email: kaduk@mit.edu